Exploiting Event-Based Communication for Real-Time Distributed and Parallel Video Content Analysis

Doctoral Dissertation by

Viktor S. Wold Eide

Submitted to the Faculty of Mathematics and Natural Sciences at the University of Oslo in partial fulfillment of the requirements for the degree Dr. Scient. in Computer Science

June 2005

To my family

Abstract

Fueled by the rapid and continuous technical advances, equipment for generating digital video is already quite inexpensive and prices are falling. Combined with progress in wireless technologies, devices with both network and video capabilities will become ubiquitous. The physical size of such video and network enabled devices is also shrinking, increasing potential usage even further. These advances create opportunities for exploiting video data from a large number of distributed devices in different kinds of applications. Example applications include systems for environmental surveillance and road traffic monitoring and regulation. Consequently, we believe that real-time video data from such distributed devices will play a central role in many different application areas, and these opportunities have motivated our research.

Clearly, a lot of applications will benefit from, or even require, an automatic approach, where computers analyze the video data in real-time. However, utilizing the potential steady flow of video data from such devices is not straight forward, and the causes of difficulties are manifold. Efficient streaming of video data from sources to a potentially large number of heterogeneous receivers is necessary, due to the massive amount of data. The computational complexity of video analysis algorithms also represents a challenge. Additionally, real-time requirements add to these challenges, as the number of calculations available for processing each video frame is limited by time. These limitations can often be reduced by distributed and parallel processing, where the analysis is done cooperatively by a number of computers. However, developing such distributed and parallel real-time applications is a complicated and time consuming process. These challenges have been addressed in our research within the domain of real-time distributed and parallel video content analysis.

The contributions of this thesis are mainly within three areas — event-based communication, video streaming, and real-time distributed and parallel video processing.

First, the thesis shows that the requirements for the targeted application domain fit well to the publish/subscribe interaction paradigm, leading to an event-based interaction model. Event-based systems differ with respect to the data model for the notifications and the expressiveness of the subscription language. We argue that content-

based event notification services are preferable, due to their expressiveness and hence flexibility. In spite of their added complexity, this thesis demonstrates that distributed content-based event notification services are both beneficial and suitable for real-time distributed and parallel video content analysis.

Second, we demonstrate the potential of exploiting such a content-based event notification service for multi-receiver video streaming. The novelty of our approach is that each video receiver is provided with independent and fine granularity selectivity along different video quality dimensions, such as region of interest, signal to noise ratio, colors, and temporal resolution. At the same time efficient delivery is maintained, in terms of network utilization and end node processing requirements. In this respect our approach represents a general and viable solution to the heterogeneity challenge in multi-receiver video streaming.

Third, with respect to real-time distributed and parallel video processing, this thesis demonstrates that the need for application level filtering and transformation of video data is reduced, by exploiting content-based event notification services for video streaming. Since different computers may subscribe to different parts of the video signal, a better match is provided between what is required and what is delivered and decoded. Hence, efficiency is improved compared to other video streaming techniques. Event-based communication provides a level of indirection and is shown to be a key factor for achieving flexible parallelization and distribution. Hence, application development is simplified and available processing resources can more easily be focused on processing bottlenecks.

The work presented in this thesis has been done within the design paradigm, that is, based on requirement analysis, specification, and design, prototypes were implemented and validated by testing. We conducted experiments for each area addressed — event-based communication, video streaming, and real-time distributed and parallel video processing. For validation purposes, the techniques from these individual areas have also been combined with state of the art techniques for distributed classification and integrated into a real-time distributed and parallel video content analysis application. The developed software has been made available as open source and as such allows others to validate the results and build upon our work.

Consequently, these results contribute to the state of the art within the areas of event-based communication and multi-receiver video streaming. Additionally, these results allow us to confidently claim that event-based communication represents a more promising foundation for the application domain of real-time distributed and parallel video content analysis, compared to other alternatives.

Acknowledgements

Although the front page seems to indicate that this thesis has been accomplished by the author individually, a lot of people have contributed to the work presented herein, in various ways.

First or all, I am deeply indebted to my supervisors, Prof. Frank Eliassen and Prof. Olav Lysne, for offering me an opportunity within the Distributed Media Journaling project. I would like to thank you for all the guiding, rewarding discussions, cooperation, encouragements, and lasting support throughout the studies. Even if the duration has been longer than originally planned, I appreciate the belief you have expressed in my ability to go the distance and for providing me with the additional required funding.

Ole-Christoffer Granmo, my fellow research scholar within the project, has also contributed to this thesis work at many different levels. We had many valuable and interesting discussions, both professionally and socially and shared office for quite some time. Fortunately, a division of research topics within the project was found, which was both complementary and based on interests. The cooperation has been fruitful and mutually improved our research results. We also had a lot of fun during late evenings and nights, working towards rapidly approaching deadlines.

Jørgen Andreas Michaelsen got involved in the project quite late. However, his contributions have been significant. Additionally, the final year of the thesis work was also a lot more enjoyable, due to interesting discussions regarding design, programming, and other more or less related topics, as well as travels and demonstrations. Thank you for your enthusiasm.

Many of the ideas presented herein evolved during discussions with the people involved in the project, and I would like to thank for this contribution. In this respect, I also would like to thank all master students involved in the project for an interesting and enjoyable time.

I also would like to thank Einar Broch Johnsen for an enjoyable time as roommates and for introducing me to the more formal aspects of computer science.

All persons associated to the Networks and Distributed Systems group, both at Simula and at the University of Oslo, have contributed to very inspiring working

conditions. This statement would still be valid when generalized to include the other departments, and in particular the system administration group at Simula in which I have been involved. Both the University and Simula have provided good working conditions. The quality and the quantity of social arrangements at Simula have been respectable and contributed significantly to the pleasant working environment.

The solid work done by the open source community is also acknowledged — in fact invaluable from a computer science research perspective. Debian GNU/Linux has provided a robust platform for research. For writing articles and the thesis, LaTeX, Emacs, and CVS have been used.

I also would like to thank, in alphabetically order, Ole-Christoffer Granmo, Arne Høstmark, Lars Olafsen, and Richard Staehli for taking the time for proofreading, in spite of all other pressing activities. It is greatly appreciated. However, any remaining mistake is my responsibility.

The sacrifices that this work has required, have been felt most strongly by my family and friends. I would like to express my gratitude to my family, in both Norway and Denmark, for the moral support and encouragements during all these years. You have been very helpful and enduring.

In particular, I would like to thank Lotte Marie for her unconditional support, when even I felt that things were out of proportions. Numerous were the times when Jakob Patrick, and lately also my twin daughters, Rebekka Viktoria and Simone Viktoria, had to accept that I was non-present, neither physically nor mentally. With respect to both family and friends, I intend to pay back some of this in the time to come. Lotte Marie and my children have also been a great source for inspiration during periods of frustration.

Viktor S. Wold Eide
June 17, 2005

Preface

The research presented in this thesis has been funded by the Norwegian Research Council through the Distributed IT Systems program, more specifically through the Distributed Media Journaling (DMJ) project, under grant no. 126103/431. Additional funding has been provided by the Department of Informatics, University of Oslo, Norway and Simula Research Laboratory, Norway.

The DMJ project was initiated as a research effort towards providing general solutions for the application domain of real-time distributed analysis of live media streams.

The project was first located at Department of Informatics, but after two years the senior researchers became affiliated to Simula Research Laboratory. Therefore, the project was relocated to this newly established research institution, which became fully operational at the end of 2001. Simula Research Laboratory performs basic research and is financed by the Norwegian government.

The senior researchers involved in the project are Prof. Frank Eliassen and Prof. Olav Lysne. Eliassen has also been the project leader. The project also financed two Dr. Scient. students, Ole-Christoffer Granmo and Viktor S. Wold Eide. Granmo defended his thesis for the degree of Dr. Scient. October 27, 2004 at the Faculty of Mathematics and Natural Sciences, University of Oslo, Norway. Over the project period, a number of master students have also been supervised in the context of the project. Some have finished, while others are at the time of writing, still active.

The overall architecture presented in this thesis was developed cooperatively by the researchers involved in the project. Within the project, Granmo has focused on issues related to classification. In particular the goal has been to provide support for controlling accuracy and timeliness in video content analysis applications. The achievements are presented collectively in his dissertation [45]. On the other hand, the research undertaken by Eide has focused on the parts related to communication, streaming, filtering/transformation, and feature extraction, or in other words, the more low-level processing performed in video content analysis applications.

In addition to the research results accomplished individually and by subsets of the researchers in the project, several efforts have been made during the project pe-

riod in order to align and integrate results. These efforts include the development of prototypes and experiments for validating the overall architecture.

Contents

Part I

Overview

Chapter 1

Introduction

After roughly three decades of exponential increase in the number of transistors which can cost effectively be integrated on a chip, mainstream computers eventually became capable of handling audio and video with quality comparable to earlier analog technologies. In the mid 1990s, 100MHz processors and gigabyte disks enabled high fidelity audio on regular computers. Some years later, around year 2000, the advent of 1GHz processors and tens of gigabytes disks enabled mainstream computers to handle video. The enablers were the advances in hardware capabilities combined with the development of compression technologies for efficient representation. At these points in time, reasonable compromises between cost, processing performance, and storage space were achieved.

The threshold for taking advantage of video information in applications is steadily decreasing. Fueled by the continuous technical advances in semiconductor technology, equipment for generating digital video is already quite inexpensive and prices are falling. Now, video cameras are commonplace in hand-held computers and mobile phones.

The technical advances have also fueled the rapid development of wireless communication technologies. These technologies are being integrated into all kinds of devices. The price also decreases continuously for computers having steadily increasing computational, storage, and communication capabilities. Consequently, devices which are capable of capturing, processing, storing, and streaming audio, video, or some other kind of sensor information are becoming ubiquitous and numerous. The physical size of such video and network enabled devices is shrinking, increasing potential usage even further.

Altogether, these advances create opportunities for exploiting the information flows from a large number of distributed devices in different kinds of applications. Such applications can be used for monitoring (e.g., environmental surveillance, traffic monitoring, or video surveillance systems) or in feedback control systems (e.g.,

for traffic regulation and intrusion detection). These opportunities have motivated our research within the domain of real-time distributed video content analysis.

However, it is not straight forward to tap into and utilize continuous flows of real-time video data from a large number of such devices. The causes of difficulties are manifold and include the inherent massive amount of video data and the computational complexity. The information embedded within audio and video data is also rather difficult to exploit directly in applications. The usefulness of a manual approach, where human beings annotate media streams with meta information, is rather limited. For a lot of applications such a manual approach is impractical or even infeasible. Consequently, a challenge is to automatically analyze media data, in order to extract relevant information.

In automatic content analysis applications, some characteristics of the media are usually first calculated. These features are then used for classification. In other words, classification is interpretation of extracted features in some application specific context. An example, which also illustrates the potential complexity of automatic content analysis, is automatic speech recognition. Such automatic audio-to-text conversion has been a topic of research for decades, but still represents a challenge. Automatic video analysis does not seem any less complicated.

Real-time requirements add to these challenges, since the number of calculations available for processing, for example each video frame, is limited by time. This limitation can often be reduced by distributed and parallel processing, where the analysis is done cooperatively by a number of computers. However, developing such distributed and parallel real-time applications is complicated, time consuming, and subsequently poses new challenges. The DMJ project was initiated as a research effort towards providing general solutions for this application domain.

1.1 Thesis Context: The DMJ Project

The work presented in this thesis was done in the context of the Distributed Media Journaling (DMJ) project. The organization of the DMJ project began in the autumn of 1998, while active research started in early 1999.

The overall goal of the DMJ project is to develop a *framework* for the *application domain* of *real-time distributed analysis* of *media streams*. Each term is described in more detail in the following.

1.1.1 Framework

The purpose of the DMJ framework is to simplify application development by providing solutions to the general issues for the targeted application domain. In other

words, the framework represents a reusable basis which can be specialized for developing specific real-time distributed content analysis applications.

1.1.2 Application Domain

Examples of applications whose development can benefit from the DMJ framework include automatic traffic surveillance systems, as described in [12]. Traffic surveillance systems may be used for traffic monitoring, electronic toll road systems, and road planning purposes. The input to such applications may be provided by mobile devices, for example attached to the vehicles, or by stationary sensors, for example located underneath or above the roads.

Similarly, environmental surveillance systems may take advantage of data provided from satellites orbiting the earth or by sensors deployed in the environment [28]. The gathered information may be used to predict extreme environmental (weather) conditions or to detect pollution.

Content analysis may also be used for control purposes. For example, in smart rooms [63], embedded sensors may detect speech, gestures, and body temperature in order to regulate room temperature and the light conditions. Another example of feedback control systems is systems, which based on video analysis, determine the traffic conditions and thereby control the entries onto highways in order to improve overall traffic throughput [24]. Such feedback control systems are real-time systems [51].

1.1.3 Real-Time

In contrast to domains where the application progress determines the rate of data consumption, i.e., off-line and batch type of processing, real-time systems must process data within certain time constraints. These constraints may specify the maximum tolerated delay between the input and the corresponding output. As an example, an upper bound may be specified on the elapsed time from a real-world event takes place and until the application is able to generate a notification about the event. This illustrates that the term “real-time” does not necessarily mean “real fast”. Rather, the term expresses that the validity of some computational result is bounded by time. In other words, the usefulness of a result is strongly dependent on when it becomes available. Due to such real-time requirements, only a limited amount of data can be buffered before timing constraints are violated. Hence, a real-time content analysis application must be able to process the data at least as fast as the data are made available.

Some real-time systems may have very strict requirements with respect to both timeliness and accuracy in order to operate safely. Systems where failure to meet

deadlines is considered a fatal fault are commonly referred to as hard real-time systems. An example is driver assists in cars for avoiding collisions. Such a safety system may determine if the braking sub-system should be activated, based on information provided by a video content analysis system constantly monitoring the scene in front of the car. Clearly, such a system must react within a fairly short time in order to be useful and be very accurate to avoid increasing the probability for accidents.

In contrast, so-called soft real-time systems tolerate some dead-line violations. The number of acceptable violations are often specified in probabilistic terms. As an example, multimedia entertainment systems have soft real-time requirements. Most people consider a few glitches acceptable when watching a movie on television or streamed over a computer network.

1.1.4 Analysis

Content analysis applications are often structured as follows. First, sensors capture information from the real world (e.g., audio or video data). The information provided by the sensors is then filtered and transformed, for example by removing certain frequencies from the audio signal or the colors in a video signal. Feature extraction algorithms, operating on filtered and transformed data streams, then calculate characteristics regarding color, texture, shape, or motion in case of video data. Finally, the extracted features are fed to classification algorithms which interpret the features in some application specific context. In a traffic surveillance application, the classification task may be to determine the registration plate numbers for cars passing a video camera. The interpretation may be both spatially (e.g., relating information within a video frame) and temporally (e.g., relating information from several consecutive video frames in order to reduce the effect of light and weather conditions). Different types of sensor information may also be combined in order to improve the accuracy. For example, multimedia analysis may take advantage of both audio and visual information, as described in [81].

The different tasks of a content analysis application, i.e., capturing, filtering/transformation, feature extraction, and classification, may each be quite complex and computationally demanding, even when applied to only a single media sample. The exponential growth in hardware capacity has been an enabler of new techniques, although better algorithms have also played an important role. In combination, faster hardware and better algorithms have enabled more analysis functions to be performed in real-time. However, there seems to be no limits to the potential complexity and hence computational demands of analysis algorithms in general.

The rate at which media samples need to be processed is clearly application dependent, as defined by the real-time requirements. Despite the tremendous improvements in hardware capabilities, real-time analysis of a single stream is potentially

computational challenging, since the number of calculations available for processing each media sample is limited by time. Even a single high quality video stream may contain massive amounts of data. Additionally, the number of streams that needs to be analyzed in real-time may be large. Consequently, a distributed approach is desirable.

1.1.5 Distribution

Some applications, such as traffic surveillance, are inherently distributed. By supporting transportation of sensor data and distributed processing, the different parts of a content analysis application may be executed at different locations, wherever most appropriate according to criteria such as efficiency, cost, and reliability. This allows for example the filtering and the feature extraction part of the analysis to be executed by a computer directly connected to a sensor device, reducing network bandwidth consumption. For sensor network applications, which are inherently distributed, this is important in order to reduce power consumption.

By supporting distributed processing, the information produced by a single sensor may also be shared by a number of different applications. As an example, the information provided by a single video camera may be used for both traffic regulation as well as for early detection of traffic accidents.

Distributed processing also allows applications to take advantage of additional processing resources. Improved performance can be achieved by parallel processing of sensor data on a number of computers. In general, parallel processing may reduce the time required to process a certain amount of data, allow more data to be processed in the same amount of time, or allow more compute intensive analysis algorithms to be used. Consequently, support for distributed and parallel processing allows different tradeoffs to be made with respect to real-time requirements, the computing environment, and the analysis algorithms.

However, developing applications for distributed and parallel processing is difficult and time consuming. Hence, framework support for distribution and parallelization is important. In order to allow application developers to focus on the application logic itself, the general issues related to distribution and parallelization should be handled by the framework.

1.1.6 Media Streams

The different application examples given so far emphasize that the DMJ framework has not been targeted at processing any particular type of information. In this respect the framework is agnostic. Media types that the framework is designed to support include audio and video, but could also include haptic data, smell, and taste. More

generally, the information could be provided by any kind of sensor device, implemented in hardware, software, or in combination.

However, the DMJ project has focused on analysis of video data for prototype implementations and the conducted experiments. The rationale has been that video represents a challenging media type, both with respect to data rates and the relatively short amount of time available between consecutive frames. As an example, television and movie quality video requires roughly 25 frames per second. In this case a new video frame is being generated every 40 milliseconds and hence the time available for processing each frame is correspondingly short. Based on this reasoning, it is likely that a framework which supports video processing applications, will also be able to support other less demanding media types.

1.2 Thesis Motivation

The motivation for the work presented in this thesis is to provide system support for the application domain of real-time distributed analysis of media streams. More specifically, the research addresses issues related to communication, streaming, filtering/transformation, and feature extraction, as described in the following.

The processing resources available from a single CPU computer are rather limited. Therefore, video content analysis applications which are unable to efficiently utilize parallel computers and/or distributed computing environments are restricted to simple video content analysis tasks, covering only a small number of video streams. Many applications within the targeted domain also have an inherent distributed nature. Additionally, harder real-time requirements may also be met by distributed and parallel processing of video streams. Altogether, these factors have motivated us to investigate distributed and parallel processing support.

The efficiency achieved by distributed and parallel processing is limited by the efficiency of the communication mechanism. Additionally, communication support is needed at different levels within the video content analysis hierarchy, supporting exchange of classification results at the highest levels and transport of media data at the lowest levels. This has motivated careful consideration of the communication mechanism efficiency.

Distributed and parallel processing may also be simplified by a communication model which allows each level of the content analysis hierarchy to be parallelized independently. In other words, changes at one level should not affect any other level, neither at compile time, start-up time, nor run-time. This has motivated our investigations of a communication mechanism which supports such separation of concerns.

Although real-time video streaming has been studied extensively for decades, transporting different parts of the video data from a source to different receivers still

represents a challenge. First and foremost, different applications may require video data from the same source. As an example, the video stream produced by a traffic surveillance camera may be used by both an application which controls the entries onto a highway and another application for detecting traffic accidents early, as mentioned previously. A video source which handles each and every receiver individually will not scale, due to for example processing, network, and power limitations. Therefore, to reduce resource consumption and hence improve scalability, packet replication should not happen at the sender side, but closer to the receivers. Secondly, a single application may utilize a number of different feature extraction algorithms which may operate on different parts of the video signal. Hence, different receivers may be interested in different parts of the video signal, spatially and/or temporally. Existing solutions to these scalability and heterogeneity challenges are too inflexible and not well suited to support parallel processing[1]. Motivated by these observations, the research topics and goals addressed by this thesis work were defined as described in the following.

1.3 Research Topics and Goals

The main question addressed by this thesis work is whether event-based communication may provide a suitable interaction mechanism for the application domain of real-time distributed and parallel video content analysis.

Event-based interaction is well recognized within the research community as a promising technology for developing loosely coupled distributed applications. Event-based communication is characterized by indirect communication, due to the lack of explicit addressing. In contrast to communication systems where forwarding is based on explicit (group) addresses, information is forwarded based on “what” is produced, rather than by “whom” and “where” the information was generated. These characteristics make event-based interaction well suited for one-to-one, one-to-many (sharing or partitioning of data), many-to-one (aggregation), and many-to-many communication. Clients connect to the event notification service and publish information in so-called event notifications or express their interest by subscribing.

Within the event-based communication paradigm there are many different variants, as described in [41]. First and foremost event-based systems differ with respect to the data model for the event notifications and the expressiveness of the subscription language. The architecture of different systems also vary a lot, ranging from centralized systems to systems which are architectured as distributed overlay networks. Consequently, different kinds of event-based interaction have to be considered.

[1] A thorough discussion of related work is given in Chapter 2, 3, and 4.

This work investigates how event-based communication can be exploited for real-time distributed and parallel video content analysis. In particular, the following goals have been identified:

- Investigate if event-based interaction is a good fit for the application domain of real-time distributed and parallel video content analysis
- Investigate if event-based communication is suitable for streaming real-time video data in particular and transporting high data rates in general
- Investigate if event-based communication can support flexible distribution and parallelization as well as efficient execution of such applications

1.4 Research Method

Computing as a discipline emerged in the early 1940s by the integration of algorithm theory, mathematical logic, and the invention of the stored-program electronic computer, as described in [31]. Hence, computing is rooted in mathematics, science, and engineering.

A framework for the discipline of computing is presented in [31]. The framework defines the three major paradigms for the discipline as *theory*, *abstraction*, and *design*. Other commonly used terms for the abstraction paradigm are *modeling* and *experimentation*. For each of these paradigms, there is an iterative process which consists of four stages.

The theory paradigm is rooted in mathematics, and the process consists of stages where: (1) the objects of study are characterized and defined, (2) relationships between these objects are hypothesized as theorems, (3) the truth of each relationship is determined by means of proofs, and (4) the results are interpreted.

The abstraction paradigm is rooted in the experimental scientific method, and the process consists of stages where: (1) a hypothesis is formed, (2) a model is constructed and predictions made, (3) experiments are designed and measurements performed, and (4) the results are analyzed.

The design paradigm is rooted in engineering. The process consists of stages where: (1) requirement analysis is performed, (2) a specification is generated based on these requirements, (3) the system is designed and implemented, and (4) the system is tested.

Unawareness of the differences between these three paradigms have caused controversies, including debates regarding which paradigm is most fundamental and people from one paradigm criticizing the work of someone in another paradigm, as described in [30]. Rather, these three paradigms are entangled and of equal importance.

The discipline of computing is a unique blend of theory, abstraction, and design, where the boundaries between theory and abstraction as well as between abstraction and design are fuzzy.

Based on the identified research topics and goals, the design paradigm has been used for the work presented in this thesis. Iteratively, requirement analysis have been performed, specifications developed, and prototypes designed and implemented. These prototypes have been used for experiments and measurements in order to validate and demonstrate the feasibility of our approach. The results have also been compared to state of the art solutions, and following prototypes have integrated the acquired and accumulated knowledge. Clearly, prototyping has played an important role for the research reported in this thesis. Such prototyping and software development is not only a product-producing activity, but also a knowledge-acquiring activity which helps reduce ignorance, as argued in [11].

The overall goal of the DMJ project has been to develop a framework for the application domain of real-time distributed analysis of media streams. In order to spot flaws and weaknesses and to report on how the solutions satisfy the identified requirements, applications have been build on top of our framework. A challenge in this respect has been to carefully select applications which are representative and which also span the potential application space of the targeted application domain.

Knowledge of previous results and state of the art has been gathered by investigating sources such as ACM Digital Library [1], CiteSeer.IST [2], the Digital Bibliography & Library Project (DBLP) [7], IEEE Xplore [5], and SpringerLink [6]. Additionally, search engines such as Google [4] have been used for discovering and crosschecking relevant information.

During the project period, research papers have been peer reviewed by experts in the field. Presentations of the research results at international conferences have also provided feedback and opportunities for exchanging ideas with other researchers. As part of our research method, software has been made available as open source in order to allow other researchers to repeat experiments and validate our results.

1.5 Unaddressed Issues

In this thesis work we have not developed new kinds of filters, transformers, or feature extractors for video analysis. Rather, the kinds used in prototypes and experiments are representative ones, and it should be rather straight forward to integrate other or new kinds into our framework.

This thesis does not address hard real-time issues. In order to provide hard real-time guarantees, resource reservation and admission control is typically required. Although hard real-time systems have been studied for a long time, their solutions have

not generally been integrated into general-purpose operating systems and networks, which is the environment that we have been working within. Therefore, we restrict the class of processing platforms considered, to general-purpose ones without special real-time processing features. Hence, the load of competing applications in the processing environment was controlled during experiments. However, we believe that the results presented in this thesis can be adapted to take advantage of processing platforms with better real-time capabilities.

Additionally, the implications of the research with respect to privacy and security issues, for example when used in surveillance applications, have been considered outside the scope of our work.

1.6 Results and Implications

The main contributions presented in this thesis have been published in a number of research papers [32–39]. The contributions are within three areas — event-based communication, video streaming, and real-time distributed video processing. The context of the research, real-time distributed and parallel video content analysis, connects these areas. This does not mean that the usefulness of the results are limited to this particular domain. On the contrary, we claim that some of the presented results may be useful in general, and not limited by the scope of the DMJ project.

The software for the content-based publish/subscribe system and video streaming has been made available as open source from [3], and as such allows others to build on our results. The following three subsections summarize the thesis contributions and are structured according to the thesis topics and goals.

1.6.1 Event-Based Communication

This thesis demonstrates that event-based communication is well suited for the domain of real-time distributed video content analysis, as argued in [34, 35]. Event-based systems differ with respect to the data model for the notifications and the expressiveness of the subscription language. Content-based event notification services offer most expressiveness and hence flexibility. In spite of their added complexity, this thesis shows that distributed content-based event notification services are both suitable and beneficial for real-time distributed video analysis.

For handling the massive amounts of data and the real-time requirements, we extended an existing distributed content-based publish/subscribe system with IP multicast support [36]. To the best of our knowledge, IP multicast support was not implemented in any other such system at that time. The system was also used experimentally for a real-time video content analysis application, as described in [39]. All

communication, even the video streaming, was handled by this distributed content-based publish/subscribe system.

This thesis has demonstrated that content-based publish/subscribe systems are well suited for the domain of real-time distributed and parallel video content analysis. Such systems offer significant advantages compared to other alternatives, including systems which use group-based communication directly. Efficient and high performance event notification services also allow content-based publish/subscribe to be used in other application areas, such as sensor networks and high performance computing.

1.6.2 Fine Granularity Multi-Receiver Video Streaming

This thesis also demonstrates that video streaming over content-based networking may provide fine granularity multi-receiver streaming. In our work content-based networking is provided by a distributed content-based event notification service. A prototype has been developed, and the video coding scheme as well as performance numbers are presented in [37, 38].

The contribution of this part of the thesis work is the bridging of techniques from the fields of video compression and streaming with content-based networking. The results include a novel video coding scheme that has been specifically developed to exploit the powerful routing capabilities of content-based networks.

In our approach, each video receiver is provided with independent and fine granularity selectivity along different video quality dimensions, such as region of interest, signal to noise ratio, colors, and temporal resolution. Efficient delivery, in terms of network utilization and end node processing requirements is maintained, as demonstrated experimentally in [38, 39].

Such fine grained selectivity is required in order to improve efficiency within the domain of real-time parallel video content analysis, where different computers process one or more video streams in parallel — functionally, spatially, and temporally.

In this thesis we argue that a video streaming scheme for handling heterogeneity in a scalable manner is also useful in general. Our approach allows video data to be streamed to a number of receivers, despite differences in network availability, end node capabilities, and receiver preferences. Consequently, our approach represents a significant step forward, compared to other approaches which use unicast or multicast directly.

1.6.3 Real-Time Distributed and Parallel Video Processing

Our video streaming scheme reduces the need for application level filtering and transformation of video data, since the filtering of event notifications is handled by the

distributed publish/subscribe system which also pushes filtering towards the source. Because the match between what is needed by different computers and what is delivered and decoded by each computer is better than with other alternatives, efficiency can be improved both network and processing wise. As shown in [39], the amount of redundant calculations can be significantly reduced.

Event-based interaction provides a level of indirection, a key factor for flexible distribution and parallelization of each logical level. In effect, the available processing resources can be focused on the processing bottlenecks at hand. This allows application development to be more decoupled from quality of service (QoS) mapping and deployment [32, 33, 39]. Consequently, these different concerns can be handled more separately.

In combination, distributed high-performance content-based publish/subscribe systems, video coding schemes which exploit the rich and powerful routing capabilities of such systems, and distributed and parallel video processing provide a promising foundation for developing efficient and scalable real-time video content-analysis applications. This was demonstrated by integrating the above presented results with the classification work done by Granmo [45], as described in [39]. Consequently, since the overhead can be kept low, harder performance requirements can be satisfied by adding computational resources, such as computers and network capacity.

1.7 Thesis Organization

The thesis is structured in two parts. Part I gives an overview of the thesis work, while Part II contains the research papers.

The rest of Part I is structured as follows. First, Chapter 2, 3, and 4 describe related work within the three main areas addressed in this thesis, i.e., event-based communication, multi-receiver video streaming, and video processing[2]. At the end of each of these three chapters we discuss some open issues which have been addressed by the work presented herein. Chapter 5 gives an overview of the research papers. The contribution of each paper is described and a comparison to related work is given at the end of the chapter. Finally, in the concluding Chapter 6 we summarize the thesis, provide some critical remarks, and present some ideas and opportunities for further work.

Part II of the thesis contains the research papers. Since each paper is self-contained, some information is necessarily repeated in different papers. In front of each paper

[2]The presentation in Chapter 2 is not limited to event-based communication only. Due to the relevance to the included research papers, the prototypes, and as potential technologies for underlying efficient event dissemination, many-to-many communication is described in general.

a separate cover page gives some information about where the paper has been published, the evaluation process and the outcome, as well as a description of the contributions made by the different authors. The papers appear in chronological order with respect to date of publication, but depending on the familiarity with the different topics the reader may choose a different reading order. The reader is encouraged to read the whole of Part I, which puts the papers into perspective.

Chapter 2

Many-to-Many Communication

In this chapter we provide background information related to many-to-many communication, important for the domain of real-time distributed video content analysis. The emphasis is on technologies suited for efficient real-time many-to-many communication. First, some background information is presented, before we proceed by describing network level multicast, application level multicast, and reliable group communication systems. Then we present Mbus and event-based communication which compared to group-based communication systems provide more flexible addressing.

2.1 Background

An important question regarding the design of communication systems is at which layer and in which entities to implement what functionality (e.g., flow control, congestion control, error control, quality of service, and multicast). In other words the fundamental question is where to put the complexity. A classic example in this respect is whether the network should be virtual circuit switched or packet switched. Closely related is the question whether the network layer should provide a connection-oriented or connectionless service to the transport layer.

In the Internet approach the functionality for flow control, congestion control, and error handling was pushed to the edges of the network — to the hosts. In this respect the Internet architecture had a clear separation between hosts, i.e., the computers, and the network realized by the routers. The argument was that applications will have to perform some of these functions themselves anyway. For some applications, such as real-time audio conferencing, network level error handling may even be disruptive, because a lost packet is regarded as better than a very late packet.

The end-to-end argument in system design, as described in [72], expresses this more clearly and advocates that functionality should only be pushed downwards in

order to significantly improve performance, otherwise it should be handled by higher levels. An advantage of this design principle is that the implementation of the network nodes and thereby the network itself is simplified and hence easier to realize and evolve.

The packet switched approach, as represented by the Internet, has been a tremendous success. The Internet has been able to support a large range of different applications — of which many were not even foreseen during the early stages of development. Although the only guarantee provided by the best-effort Internet model is that each router forwards a packet towards the destination with a probability larger than zero, the service quality experienced is usually quite good, due to traffic provisioning and congestion control. Multi-point communication requirements can also be reasonably supported for applications which have non real-time and modest bandwidth requirements. A straight forward although inefficient solution, is to utilize underlying point-to-point communication to mimic multicast behavior (e.g., as done for email delivery).

The best-effort approach taken in the Internet has been less successful for handling applications with real-time requirements. Only recently has the Internet been used to some extent for telephony. This is an application with fairly strict requirements regarding delay, but very modest requirements for bandwidth. On the order of 10kbps is required for transporting voice data of reasonable quality. Requirements from other interactive applications, such as web browsing, have driven bandwidth availability far beyond these modest requirements. The probability of successfully making phone calls over the Internet seems rather good and Internet telephony has become increasingly common.

Other application domains, such as interactive multi-user audio-video conferencing and real-time distributed video content analysis, combine real-time requirements with requirements for multi-point communication and large amounts of bandwidth. The insufficient support for applications having such requirements has been recognized for decades, and in the following some representative approaches are presented.

2.2 Network Layer Multicast

In order to handle multicast efficiently, extensions to the Internet network layer were proposed in the late 1980s, as described in [29]. The motivation for pushing multicast functionality into the network nodes was to reduce link stress and delay to a bare minimum. The link stress denotes the number of duplicate packets traversing a single link. In other words the rationale for extending the Internet network layer with multicast functionality was to improve performance significantly, which is in line with the design principle expressed in the end-to-end argument [72].

IP multicast is designed in such a way that each multicast packet should traverse each network link at most once, and the path taken by the packet should be close to optimal. Hence, both link stress and delay should be close to minimum. For the links close to the source this is important, especially when considering potential large multicast groups. Since the source host only has to send one instance of a packet, and not one packet to each receiver, delay and processing resource consumption are also reduced. Without multicast support, a host may have to send several copies of the same packet, and hence the last copy leaving the host is delayed. In IP multicast the packet replication and forwarding task is removed from the source host and is instead handled by the routers. The routers replicate packets close to the destination nodes and in parallel.

In addition to replication and forwarding of packets, IP multicast performs group membership management and multicast routing, i.e., maintaining data delivery paths. In the IP multicast model, a portion of the IP address space is used as multicast addresses, and each address identifies an IP multicast group. Hosts may join and leave groups using an implementation of the Internet Group Management Protocol (RFC 3376).

Fueled by the interest in the research communities for a multicast capable network infrastructure, the Multicast Backbone (MBone) emerged in the early 1990s. Mbone was initially constructed as an overlay network layered atop of the Internet, where routing and forwarding was handled by workstations running multicast routing software, as described in [8].

Since then, different IP multicast routing protocols have been developed for both intradomain and interdomain multicast routing [8]. In essence, the routers exchange information in order to build and maintain the multicast distribution trees for different groups, and routers maintain per group state in order to determine if a packet must be replicated and on which interfaces to forward these duplicated packets.

By pushing functionality down and implementing multicast at the IP layer, complexity and cost has been added at the Internet network level. In spite of roughly fifteen years of massive and joint efforts by academia, standardization organizations, research institutions, and commercial vendors, IP multicast has still not matured and become a ubiquitous service on which application developers can rely. IP multicast still faces many challenges in regard to scalability, security, deployment, and management issues.

2.3 Application Layer Multicast

In response to the lack of ubiquitous multicast support in the Internet, researchers have begun to question whether IP multicast will become a true Internet service and

whether multicast support at the network level is practically possible. Anyhow, there was a need for intermediate solutions. A survey of different proposals for an alternative group communication service is provided in [40]. In the following we describe some approaches.

In the early 2000s research proposals for implementing multicast functionality in the end systems emerged, as described in [25]. In these overlay multicast networks all multicast functionality, including group membership control, routing, and packet forwarding, is implemented exclusively in the hosts. Consequently, routers only need to support IP unicast traffic. The approaches differ, where some require infrastructure support, while others leverage on recent developments in peer-to-peer technologies and are completely decentralized dynamically constructed and maintained overlay networks. One advantage of application layer overlays is the opportunity for supporting application specific customization of the overlay network. Shortly, we will describe some approaches for application layer multicast, but first some background information on peer-to-peer technologies is provided.

Peer-to-peer technologies have since the late 1990s received widespread attention and are now extensively being used to build distributed overlay networks [27][1]. These overlay networks try to take advantage of and harvest the additional resources represented by each computer joining the overlay. Each peer provides access to some of its underlying computer resources, and hence plays both the resource provider and the resource consumer roles. In effect, peer-to-peer systems seek to spread out resource consumption throughout the entire overlay network and compared to client-server systems the communication pattern appears more symmetric.

Structured peer-to-peer systems, such as CAN [68] and Pastry [70], implement distributed hash tables (DHTs) which provide distributed content location and routing. Each node which joins the overlay network becomes responsible for a part of the key space and holds the values associated to these keys, i.e., (key,value) pairs. Whenever a message is sent, a key must be provided which determines the destination node for the message. The key space and the routing algorithms are constructed in such a way that each message is routed towards the node which is responsible for the particular key, and where the number of routing hops is logarithmic in the number of nodes in the overlay. The algorithms only assume local knowledge in each node, which results in good scaling properties. Thereby, a lookup message is routed within the overlay to the node which is responsible for the key, in a few hops. The resulting dynamically constructed self-organized multi-hop overlay networks operate

[1]One may argue that the routing protocols used to construct and update routing tables in the Internet are examples of peer-to-peer communication. The Internet was also overlayed on top of, e.g., telephone lines. Similarly, the network-news transport protocol (NNTP) also resembles peer-to-peer communication. Each host on the Internet was also accessible from any other host and could play both a server and a client role simultaneously.

at the application layer and do topology construction, routing and data forwarding. The routing algorithms for different peer-to-peer systems differ. CAN takes advantage of a Cartesian hyperspace for routing, while Pastry conceptually organizes the keys and nodes in a circular space. Such systems assume point-to-point connections between hosts, although packets in reality typically makes a number of hops on both the network and the link layer. For efficiency, both Pastry and CAN are able to take network locality into consideration.

Peer-to-peer systems have been used for building application layer overlay multicast networks, and examples include CAN-multicast [69], which is built on top of CAN, and Scribe [20], which is built on top of Pastry. In CAN-multicast a separate overlay is built for each multicast group, and multicast messages are broadcasted/flooded within that overlay. An advantage is that only nodes which are members of a multicast group contribute. Additionally, due to the flooding approach there is no single multicast distribution tree for the entire group. The disadvantage is the cost of building and maintaining a separate overlay per group.

Scribe uses a rendezvous approach, where the hash of the group name gives a key which identifies the rendezvous node. Nodes interested in receiving multicast traffic send join messages towards the rendezvous node. The join messages follow the Pastry routes to the rendezvous node and intermediate nodes update table entries which reflect downstream interested clients. Clients send multicast traffic to the rendezvous node, which forwards the message on the multicast tree formed by the reverse path of the join requests. This receiver-driven approach does not require a separate overlay per group, but nodes may have to contribute resources to multicast groups which they have no interest in. Additionally, all multicast traffic for the group flows through the rendezvous node.

Similar to IP multicast, reducing the link stress and delay are some of the most important challenges. The baseline for comparing the link stress for different application layer multicast approaches is IP multicast. The worst case on the other hand is represented by naively using unicast to mimic multicast. Similarly, the delay from source to receiver in the overlay is typically compared to the delay that would have been experienced with unicast in the underlying physical network. Performance numbers for CAN-multicast and Scribe are presented in [20, 69], and a performance comparison between these two systems is provided in [21].

2.4 Reliable Group Communication

Lack of group communication support in the IP communication suite inspired a number of research efforts in the 1980s. In addition to research on IP multicast these efforts lead to technologies for reliable group and multicast communication.

The distributed computing model described in [13] is based on process groups with stronger semantics regarding error handling. This model is commonly referred to as reliable process group computing or virtual synchronous group computing. In this model processes may join and leave groups, and the system informs each processes about other processes joining and leaving the group. The system automatically manages group membership, synchronized with multicast sending and delivery. Different ordering guarantees are provided for delivering multicast messages to group members, such as totally ordered and causally ordered multicast. State can also be transferred within the group, where the state transfer appears atomic with respect to the change in membership. This is useful for initializing processes which join the group and for handling situations where processes leave the group or crash. The model has been implemented in different systems, including Isis, Horus, and Ensemble, and used in a wide range of application domains [13].

The Scalable Reliable Multicast (SRM) protocol [42] is designed to provide applications with flexibility and functionality such that application specific requirements may be taken into account. According to this philosophy additional layers built atop of SRM may provide stronger ordering guarantees, if and when needed. Consequently, applications which only require reliable multicast delivery will not have to pay the price associated with totally or causally order delivery. A major concern of SRM is to achieve scalability by reducing the number of requests for repair, the number of repair messages, and the delay introduced during loss recovery.

Taking a step back, the price paid for providing end-to-end reliability for one-to-one communication is delay, due to potential retransmissions. Reliable many-to-many communication is much more complex. The combination of many-to-many communication, reliability, consistency, scalability, and real-time is very challenging. Handling this problem is increasingly difficult as the system scales in the number of participants and/or the number of messages exchanged per second. As explained in [14] the performance of protocols for reliable group communication degrades dramatically when exposed to ordinary transient problems, such as processor and network scheduling delays and packet losses. The authors argue that even a single perturbed process impacts and slows down the whole group. They also argue that this is the case for multicast protocols having weaker reliability goals, such as SRM. Hence, it seems very difficult to avoid throughput instability in reliable group communication systems.

An approach for reducing these scalability problems, advocated by the authors of [14], are protocols which employ epidemic-style/gossip-based algorithms. The authors claim that such probabilistic protocols seem to be better at rinding out instability caused by infrastructure and host problems and thereby provide gracefully degradation, instead of collapse.

2.5 Mbus

As described in [62], Mbus (RFC 3259) is a software bus designed to support coordination and control between different application entities. The Mbus protocol specification defines message addressing, transport, security issues, and message syntax for a lightweight message oriented infrastructure for ad-hoc composition of heterogeneous components. As stated in [62], Mbus is not intended for use as a wide area conference control protocol, for security, scalability, message reliability, and delay reasons. Mbus is designed for intra host and inter host usage and exploits IP multicast for efficient message dissemination.

Mbus supports binding of different causalities by using a "broadcasting" and filtering technique. All components participating in a specific Mbus session subscribe to an IP multicast address, and in effect Mbus messages are "broadcasted" to the set of computers hosting components participating in this Mbus session. Mbus is implemented as a library which is linked into the applications. Hence, the Mbus layer in each component sees all Mbus traffic and must filter and discard unwanted messages.

Important for message selection and filtering is the addressing used in Mbus. The address of a component is specified when initializing the Mbus layer, and selection and filtering is performed based on this address. The Mbus header includes source and destination addresses. Each address is a sequence of attribute-value pairs, of which exactly one pair is guaranteed to be unique, i.e., the combination of process identifier, process demultiplexer, and IP address. Each Mbus component receives messages addressed to any subset of its own address. A Mbus component is able to address a single, *"(id:7-1@129.240.64.28)"*, a subset, *"(media_type:video component_type:E)"*, or all, *"()"*, Mbus components by specifying an appropriate sequence of attribute-value pairs.

The Mbus acts as a layer of indirection between components, giving both access and location transparency. Component awareness is supported by a soft-state approach, where the Mbus layer listens and periodically sends self announcements messages on behalf of its component.

Regarding scalability, message propagation delay, and reliability of message delivery, Mbus inherits many of its characteristics from IP multicast, which is realized as a distributed and scalable service. The Mbus component awareness functionality limits scalability, but the rate of self announcements is adapted to the number of entities participating in a session.

At the transport level, Mbus messages are encapsulated in UDP packets and transported unreliably by IP multicast. In the special case where the message is targeted at exactly one receiver, reliable unicast delivery is supported by the Mbus layer, using acknowledgement, timeout, and retransmissions mechanisms. The Mbus/UDP/IP multicast protocol stack does not give any ordering guarantees.

2.6 Event-Based Communication

Despite tremendous success and widespread usage of the client/server interaction model, a large class of distributed applications are better structured as a number of asynchronously processing and communicating entities. Such applications fit well to the publish/subscribe interaction model. Event-based interaction provides a number of distinguishing characteristics, such as asynchronous communication, lack of explicit addressing, and loose coupling. The communication is indirect and decoupled with respect to time, space, and synchronization. Altogether, these characteristics make event-based communication suitable for applications having a variety of one-to-one, one-to-many (sharing or partitioning of data), many-to-one (aggregation), and many-to-many communication patterns. A survey of the publish/subscribe communication paradigm and the relations to other interaction paradigms is given in [41].

Event-based systems rely on some kind of event notification service. The architecture of such services vary a lot, ranging from centralized systems to systems which are architectured as distributed overlay networks. A distributed event notification service is realized by a number of cooperating servers, also denoted *brokers* in the literature. For distributed services, servers may be interconnected in different topologies, for example trees, directed acyclic graphs, general graphs, or hybrid variants, as described in [18].

Clients connect to these servers and are either *objects of interest*, *interested parties*, or both. An object of interest publishes event notifications, or just notifications for short, while interested parties subscribe in order to express interest in particular notifications. The responsibility of the event notification service is routing and forwarding of notifications from objects of interest to interested parties. In essence, the servers jointly form an overlay network of notification routers.

Event-based systems differ with respect to the data model for the event notifications and the expressiveness of the subscription language. The difference between such systems can be characterized by what part of the data model is made visible to the event notification service in subscriptions. A subscription language with fine grained expressiveness allows the service to filter notifications early, but adds implementation complexity and run-time overhead for evaluating notifications against subscriptions. Consequently, expressiveness and scalability are conflicting and have to be balanced.

In *channel-based* systems, e.g. as specified by the CORBA Event Service [57], an interested party may subscribe to a channel and in effect receive all notifications sent across that particular channel. These systems are often also referred to as *topic-based* or *group-based* systems. Scribe [20], which has been proposed as an application level multicast infrastructure, has also been referred to as a topic-based event notification infrastructure [71]. In such systems only a channel, topic, or group identifier is

exposed to the event notification service in subscriptions. Consequently, the expressiveness is severely limited and more extensive client-side filtering is required.

Subject-based systems, such as TIBCO Rendezvous [80], provide somewhat finer granularity with respect to selection of notifications. Each notification contains a well-known *subject* attribute, and interested parties may register interest by specifying an expression which will be evaluated against the value of this subject attribute. Subject-based systems may also support hierarchical subject names and/or wild-card expressions on subject identifiers to further improve the expressiveness of subscriptions. The object of interest determines the most appropriate subject for each notification published.

Content-based systems, such as Elvin [73], Gryphon [61], Hermes [65], and Siena [18], provide even finer granularity. In such systems notifications typically consist of a number of attribute/value pairs. A subscription may include an arbitrary number of attribute names and filtering criteria on their values. Hence, content-based systems increase subscription selectivity by allowing subscriptions along multiple dimensions.

Distributed content-based publish/subscribe systems are often architectured to operate over wide area networks (WAN), for example private networks or the Internet. The construction and maintenance of networks for distributed content-based publish/subscribe systems are challenging. In [19] a routing scheme for content-based networking is proposed which uses a combination of a traditional broadcast protocol and a content-based routing protocol. Not surprisingly, peer-to-peer technologies have recently also been exploited for constructing content-based publish/subscribe overlay networks, as described in [64, 78]. The approach described in [78] uses subscriptions to prune the flow of notifications, while also advertisement filters are used in [64] to reduce the flow of subscriptions. Simulation results presented in [64] show improved routing efficiency and reduced amount of state in each node, compared to standard content-based publish/subscribe systems. In [78] analytical results are presented which show the advantage of utilizing peer-to-peer systems for building content-based publish/subscribe systems. These early research efforts seem promising regarding automatic construction, self configuration, and adaptation of content-based publish/subscribe overlay networks.

2.7 Discussion

With respect to expressiveness, multicast and group-based systems provide only the ability to join and leave groups. Regardless of being implemented at the network or the application layer, such systems perform routing and data dissemination based on these group memberships. Similarly, channel-based event notification services only allow joining or leaving different channels. Hence, these systems provide a rather

course grained way of expressing interest, and may therefore increase the need for end system filtering. In order to improve the selectivity and the match between what is of interest and what is delivered, a number of groups can be used. A scheme is then required which specifies which groups should receive which information. Determining such a scheme statically is difficult, but more problematic is the explosion in the number of groups necessary for providing fine grained selectivity. Additionally, if such a mapping is exposed to and used directly in applications, the mapping becomes static. In comparison the addressing provided by Mbus is more flexible, although delivery is determined by the address assigned to a component. Content-based event notification services provide even more expressiveness and thereby improved selectivity. Considering expressiveness and selectivity, a content-based publish/subscribe system can trivially implement the other systems described in this chapter, while the opposite is not the case.

For real-time distributed video content analysis the performance of the communication system is also a major concern. Therefore, efficient use of underlying communication primitives is required. IP multicast maps well onto layer two multicast, is intended as a global and distributed service, and provides good performance. On the other hand, it is not ubiquitous and only best-effort delivery is provided. As a result IP multicast is currently not very attractive for wide area usage, such as interdomain or Internet usage. On the other hand, using IP multicast in the LAN/intradomain case is reasonable, as the opportunities for controlling both network equipment and traffic are much better. For application layer multicast the situation is almost the opposite. Group-based communication on top of unicast may be reasonable in the WAN/Internet case, but for the LAN/intradomain case, where multicast is often supported in switches and routers, the performance potential is wasted when multicast is realized by underlying point-to-point communication. Reliable group communication may be highly beneficial for the signaling and control plane of distributed real-time video content analysis applications. However, these systems seem less appropriate for the data flow plane, as a single perturbed process may slow down the application to a point where real-time requirements are violated. Mbus is suitable for transporting data within a LAN or a host, but the end node filtering approach makes Mbus inadequate for handling huge amounts of data (e.g., for streaming several videos concurrently). For distributed event notification services, much effort has been devoted to the design and architecture of services for WAN/Internet usage. With respect to performance, a main concern is how to efficiently distribute event notifications between the servers which cooperatively realize the distributed service. The challenge of utilizing multicast communication for efficient event notification dissemination in content-based publish/subscribe systems is well known [18, 41]. This issue has been studied in [46, 61], but is not implemented in any system that we are aware of.

The expressiveness of content-based event notification services and the performance potential represented by native multicast support have motivated our work on bringing these technologies together. Combined, the expressiveness and the performance of such a service may provide a foundation for fine-grained multi-receiver video streaming in particular and transporting high data rates in general, as shown in the following chapters.

Chapter 3

Multi-Receiver Video Streaming

The advances within the fields of network level communication technologies, transport protocols, and higher level communication systems have allowed and influenced the development of new approaches for transporting and delivering real-time video data over computer networks. In this chapter, we present some background information regarding video coding and real-time multi-receiver video streaming. First, we briefly provide some background information on video coding, as an introduction to the challenges of multi-receiver video streaming. Then, some different approaches for handling these challenges are presented.

3.1 Background

What fundamentally distinguishes video compression from compression of still images, is the ability to exploit the temporal redundancy between consecutive frames. This research area dates back to the 1960s, as described in [75]. The increase in available processing capacity has allowed more complex, but also more powerful, techniques to be used in order to realize more efficient video compression schemes. The research and development conducted by research groups, standardization organizations, and commercial companies have given rise to a multitude of different video compression techniques and formats.

The latest standard developed jointly by ITU and ISO is the H.264/AVC standard. An overview of video coding concepts, some historical information, as well a description of the techniques used in H.264/AVC, is given in [75]. Compared to earlier standards, the H.264/AVC standard achieves significantly improved compression efficiency. Numbers reported state approximately 50% savings in bit rates for the same perceptual quality. This efficiency gain is mostly due to improvements in the motion compensation prediction, but comes as a high computational cost.

Efficient delivery of video data over the Internet has been studied extensively for decades. As a result relatively good solutions for point-to-point video streaming exist.

Video multicast within large LANs is challenging, as described by the authors of [74]. The main challenges are related to heterogeneity and efficient delivery issues. Real-time multi-receiver video streaming over the Internet is even more complicated, as pointed out in different survey articles [43, 52]. Despite extensive research, real-time multi-receiver video streaming still represents a challenge which awaits satisfactory solutions. One of the main sources for the difficulties is heterogeneity, as discussed in the following.

3.1.1 Heterogeneity Challenges

Video servers and clients are connected to the network by diverse technologies, which again may have rather different characteristics. Similarly, the end node capabilities may differ radically with respect to processing capabilities, display resolution, and power availability. The resource availability may also change over time. As an example, the available bandwidth may vary due to transient network failures, congestion, and changes in signal to noise ratio for mobile and wireless equipment.

Additionally, different receivers may have different preferences with respect to the importance of the different video quality dimensions. This adds to the heterogeneity challenge. In situations when resources become scarce, different receivers have different preferences with respect to adaptation. As an example, some clients may prefer a decrease in temporal resolution instead of a decrease in spatial resolution, and vice versa.

The above reasoning illustrates that handling heterogeneity is somewhat related to handling adaptation, although at a different time scale. Both heterogeneity and adaptation necessitates systems capable of operating in a range of different circumstances. Consequently, the challenge is to provide each video receiver with the best possible video quality when considering resource limitations and preferences, and at the same time to maintain efficiency and scalability in the network and the end systems.

3.1.2 Efficient Delivery Challenges

Unicast delivery, where clients connect directly to a server, may provide each client with a customized stream. Such a stream may be specifically generated in order to fit the capabilities of the receiving node as well as receiver preferences. In order to cope with variations in available bandwidth, the stream may also be adapted during runtime. Such point-to-point streaming simplifies adaptation, as other streams may not even be taken into consideration.

For streaming, standards for encapsulating and transporting the compressed video data over networks, are needed. As an example, in addition to the video coding layer specified in the H.264/AVC standard, a network abstraction layer is also specified for packaging the encoded video for network transport. For streaming over the Internet, the Real-Time Protocol (RTP) (RFC 3550) is often used. RTP requires an additional specification for each different video payload format. A RTP payload format for H.264 video has just recently become an Internet standard (RFC 3984). Such technologies provide relatively good solutions for point-to-point video streaming. However, delivering the same stream to a number of receivers directly over unicast is neither an efficient nor a scalable solution. The processing capacity of the server or the network links close to the server are easily overloaded when the number of video clients increases. Consequently, scalability is severely limited when a number of unicast streams are used to achieve multicast behavior.

Multicast delivery, provided at the network layer or by overlay networks, may improve network efficiency. However, a single multicast stream is not well suited for handling the heterogeneity challenge. A single multicast stream provides very few options. Some of the incoming packets may be discarded in order to reduce processing requirements, but network bandwidth requirements are not affected.

In *simulcast delivery*, the same video content may be streamed over a number of separate multicast channels, and each stream provides a different tradeoff between quality characteristics and resource requirements. Consequently, each video receiver is provided with a choice between a few streams and may subscribe to the multicast address which carries the video stream which best matches resource requirements and preferences. Video receivers may then adapt by joining another and more appropriate stream. The disadvantage of the simulcast approach is the inefficiency caused by redundant video information being sent on different multicast channels. The inefficiency is amplified when a large number of different streams are provided in order to closely match the preferences of each receiver.

Clearly, the challenge is to provide each video client with good quality, while maintaining efficiency in terms of network and processing resource consumption.

3.2 Layered Video Coding and Multicast

An approach for handling both the multi-receiver and the heterogeneity challenge is to combine layered video coding techniques with transport over several multicast channels. A pioneering work in this respect is video coding for receiver-driven layered multicast, proposed in [56]. In the presented approach the video signal is encoded into a number of layers. The layers are coded cumulatively in order to reduce the amount of redundant information across layers. Hence, the base layer provides the

lowest quality, while each additional layer improves the video quality. The scheme described in [56] supports spatial and temporal scalability.

For transport the compressed video data are encapsulated into packets and destined to the IP multicast address for the particular layer. Each of these layers is then transported over the network on a separate IP multicast channel. The forwarding and late replication of packets is handled in the network by IP multicast. Video receivers may then subscribe to one or more of these multicast channels. Clients with low bandwidth connections may subscribe only to the IP multicast channel carrying the base layer information. Other clients, having better network connections, may subscribe to a number of additional multicast addresses, as available bandwidth permits. Upon detecting an overload situation, a receiver may leave the IP multicast group carrying the topmost refinement layer and thereby reduce resource requirements.

An advantage of this scheme is that the sender does not have to be involved in the adaptation process. Receivers perform join-experiments to determine the maximum available bandwidth and leave groups upon detecting congestion. Consequently, the level of indirection provided by IP multicast and this receiver driven approach provide scalability. However, in order to reduce oscillation effects and receivers joining and leaving IP multicast groups to probe the available bandwidth, an algorithm has been implemented where receivers learn about other receivers failed join-experiments.

A similar approach which support color and resolution scalability is presented in [79]. In this approach the luminance and the chrominance part of the video signal are handled separately. Three resolution layers are generated for both the luminance and the chrominance part, resulting in a single base layer and five enhancement layers. The described scheme does not support inter frame coding, and temporal scalability is supported only by changing the frame rate in the system.

With respect to user preferences, a layering policy determines the dimension to be refined for each additional layer in [56]. This layering policy is fixed at the sender side and determines how the quality of the video is refined when a video receiver subscribes to an additional IP multicast address. As an example, the first additional layer may improve the spatial quality, while the second one improves the frame rate. In other words, the receivers must live with the policy specified by the sender. In [79] receivers may independently select resolution and whether color information should be included.

3.3 Priority-Progress Multicast

In [50] a framework for real-time quality-adaptive media streaming is presented. The motivation for the work is to handle the variability in available bandwidth in best-effort networks, such as the Internet. Additionally, efficient coding of video data

introduces variability in bit rate requirements, due to the video content itself. The goal is to handle such fluctuations and allow video data to be encoded once and streamed anywhere, by adapting to bandwidth availability.

In the described scheme, called priority-progress streaming, video data are transformed into a scalable representation which supports spatial and temporal scalability. Spatial scalability is realized by transcoding DCT (Discrete Cosine Transform) coefficients hierarchically to a set of levels, while temporal scalability is realized by frame dropping. This allows video data to be broken up into small chunks of application level data units. These chunks can then be assigned different priorities based on some policy. The described approach relies on utility functions for expressing the importance of the quality along and between different video quality dimensions. The timeline is divided into distinct time intervals, so-called adaptation windows. During streaming, adaptation is handled by priority dropping. The idea is that the most important data are sent first. Whatever remains in the previous adaptation window is then dropped when the system transits from one adaptation window to the next.

For one-to-many streaming an overlay multicast structure is used. Each edge in the overlay distribution tree is realized by means of the priority-progress unicast streaming approach. The described implementation utilizes TCP for transport for both unicast and multicast delivery. As a result, the congestion control is TCP friendly. In contrast to the unicast case where dropping only happens at the sender side, dropping may happen at any node within the multicast distribution tree. Hence, different parts of the distribution tree may have different data rates, and different clients may receive the video data in different quality. From our understanding only a single policy may be specified for a single session, since the priorities are assigned by the priority mapper, which is co-located with the sender at the root of the tree.

3.4 Media Streaming over the CORBA Event Service

The CORBA Event Service was originally not meant for handling real-time multimedia data. This has led some researchers to propose extensions to the CORBA Event Service, as described in [22, 23, 66].

In [22, 23] the authors suggest using an event service for transporting potentially large event notifications, carrying multimedia content. The enhancements to the CORBA Event Service add support for specification of QoS, security mechanisms, and event-based multimedia streaming. The authors introduce the concept of *stream events*, i.e., events that encapsulate media content. In other words, each such stream event encapsulates a fragment of a continuous multimedia stream. In addition to the standard CORBA Event Service channel type, the authors propose multicast and stream type channels. Multicast channels use a reliable multicast protocol for

optimizing the event delivery mechanism. A stream type event channel distributes multimedia data flows and also utilizes multicast for data delivery.

To better cope with the real-time characteristics of multimedia, event channel creators and users may specify event channel QoS properties. These properties determine for example reliability of event delivery (best-effort or try *n* times) and the scheduling priority of event channel threads. The reliable multicast protocol described in [23] exploits IP multicast for efficient event dissemination. For loss recovery a negative acknowledgement scheme (e.g., running on top of TCP) is used for requesting retransmissions. The event-channel responds to a retransmit request by resending the missing event and all later events, also by multicast. The authors argue that their scheme is reasonable due to its simplicity, that most resend requests involve a large percentage of the receivers, and that usually a number of consecutive events are lost.

Different test applications have been developed on top of the enhanced CORBA Event Service implementation. The applications described in [22, 23] include video streaming and a multichannel Internet radio service. On the receiver side a player implemented on top of the Java Media Framework was used for presenting the media data received over the event channel.

3.5 Media Gateway Systems

As another solution to the heterogeneity problems associated with multi-receiver media streaming, media gateway systems have been proposed [60, 67]. These systems are overlay networks — the media gateways are the internal network nodes, while senders and receivers are at the edges. Gateways receive media streams from upstream nodes, before forwarding the processed and potentially transformed streams to downstream gateways and receivers. In other words the gateways rely on some kind of active network nodes, where the processing is domain specific.

Examples of processing include transformations which reduce the bandwidth of a stream by changing the quality in dimensions such as frame rate (temporal scaling), frame size (resolution scaling), and quality (signal to noise ratio scaling). Other transformations may involve changing the media format, for example from MPEG-4 to H.263. Additionally, more complex operations may generate new streams, based on a number of other streams. Examples include the ability to create picture-in-picture effects and scaling down a number of video streams before composing these scaled down streams into an “overview” video stream. Consequently, such transformations allow media gateway systems to bridge the heterogeneity gap created by differences in resource availability, hardware capabilities, and software incompatibilities.

Constructing such overlay networks is challenging and involves determining the kind of media processing performed at each overlay node. The construction may be

driven by goals such as minimizing the average or maximum delay experienced by receivers or minimizing the overall processing or bandwidth consumption. The problem is further complicated when considering combinations, such as reducing both bandwidth consumption and maximum delay. Additionally, the dynamic nature of media gateway systems have to be taken into account, because receivers may join and leave at any time.

The media processing computations described in [60] are represented by small scripts. The paper addresses the problem of how to decompose such computations into a number of subcomputations and the mapping of these subcomputations onto gateways. The goal of the described system is to reduce the bandwidth consumption. The media gateway system takes care of construction and maintenance of the overlay network in order to achieve this goal.

3.6 Discussion

Clearly, the advances in communication technologies have influenced the development of techniques for multi-receiver video streaming. The different approaches described in this chapter take advantage of IP multicast communication, overlay distribution networks, and channel-based event notification services.

The combination of layered video coding and multicast provides an interesting solution for scalable multi-receiver video streaming. The presented approaches address both the layered encoding problem and the layered transmission problem, in combination. However, some receivers may prefer temporal resolution over spatial quality, and vice versa. A problem with having the layering policy fixed at the sender side is such conflicting user preferences. In effect, video receivers become unable to customize the video stream independently. In the presented approaches only two quality dimensions are considered. Specifying such a layering policy becomes even more problematic as the number of scalable video quality dimensions increases. It also seems difficult to extend the proposed schemes for supporting more dimensions, without utilizing a very large number of multicast addresses.

Priority-progress multicast represents a promising approach for adaptive multi-receiver video streaming in best-effort computer network environments. The priorities are assigned to the different application level data units by the priority mapper at the sender side. Consequently, the system seems unable to handle video receivers with conflicting preferences regarding the relative importance of the different video quality dimensions.

A video streaming solution for channel-based event notification services only provides clients with the ability to join or leave a channel, and thereby the stream. Each event notification is forwarded within the channel and not routed independently. The

lack of expressiveness in channel-based event notification services makes these systems inadequate for providing fine granularity multi-receiver video streaming.

A media gateway may perform any kind of video filtering and transformation operation, and stream customization is thereby supported. As an example, a gateway may partition a video stream spatially as a preparation step for parallel processing. The cost associated with this flexibility is increased network and processing resource consumption and additional delay. Although several receivers may share a single gateway, they may share interest in only some part of the video signal. Hence, it seems difficult to handle such cases efficiently, processing and delivery wise.

Consequently, none of these systems seem to allow different receivers to independently customize the video stream along several different video quality dimensions. A solution which bridges video coding and fine granularity multi-receiver streaming is therefore desired. Such a solution should provide efficient one-to-many transport and delivery. These observations have motivated our research on video streaming over content-based event notification services.

Chapter 4

Video Processing

Over the years, many systems have been developed in order to ease the development of video processing applications. Due to the large number of such systems, the presentation in this chapter is not intended to be exhaustive. Rather, some representative approaches are presented.

4.1 Background

Developing multimedia applications from scratch is very time consuming. Efficient representation is necessary for multimedia data in general and video data in particular in order to reduce storage space and network bandwidth requirements. Developing software for manipulating these often complex formats is challenging. Consequently, a number of systems, including libraries and frameworks, have been developed over the time in order to simplify multimedia application development. These systems have been developed by both research institutions, commercial vendors, and industry consortia. The systems differ in scope, where some target multimedia application development in general and include support for video coding and streaming, while others have concentrated solely on the video processing part.

Several commercial frameworks exist, including the Open Source Computer Vision Library (OpenCV) [47] developed by Intel, OpenML [49] from the Khronos Group industry consortium, the Java Media Framework (JMF) [76] from Sun Microsystems, and DirectShow [16] from Microsoft. OpenML is rather new, and the first OpenML software development kit became available in April 2004. The functionality provided by DirectShow and JMF are similar. The following description of commercial frameworks is therefore limited to the OpenCV library and the Java Media Framework. We first describe these two commercially initiated projects, before some systems developed by different research communities are presented.

4.2 OpenCV

As described in [17, 47], the Open Source Computer Vision library (OpenCV) effort was initiated by Intel. The OpenCV library was released as an Alpha version in year 2000, Beta in 2003, and an official 1.0 release is expected in 2005. The library is mainly intended for developing real-time computer vision applications for human-computer interaction, robotics, and surveillance. OpenCV includes a collection of functions for a number of different areas, including motion analysis and object tracking, image analysis, structural analysis, image recognition, and 3D reconstruction. Typically, different functions from each of these different areas are combined in order to develop computer vision applications.

Closely related to the OpenCV library is another library from Intel, the so-called Integrated Performance Primitives (IPP) library. The IPP library exploits specific hardware instructions in Intel processors for improved performance. Compared to OpenCV the functionality provided by the IPP library is more low-level. The library provides functions for signal, image, and video processing. If present, the OpenCV library is able to take advantage of the IPP library for improved performance, which is important for the targeted application domain.

4.3 Java Media Framework

As described in [76], the motivation for developing the Java Media Framework (JMF) was to simplify incorporation of time-based media, such as audio and video, into Java applications and Java applets. The first version of JMF was developed cooperatively by Sun Microsystems, Intel Corporation, and Silicon Graphics and was targeted primarily at playback and presentations. In late 1999, the JMF 2.0 API was released in a joint effort by Sun Microsystems and IBM Corporation, adding support for streaming, capturing, and storing of media data.

In this latest version, JMF performs low-level media tasks, such as capture, transport, streaming, (de)multiplexing, (de)coding, and rendering. The reference implementation of JMF from Sun and IBM provides support for some standard file formats, video and audio compression schemes, as well as RTP-based unicast and multicast streaming protocols.

The extensibility of the framework is provided by a plug-in architecture. The supported plug-in types are demultiplexers, codecs, effects, multiplexers, and renderers. Thereby, JMF provides access to the raw media data and allows integration of customized media processing algorithms. This allows third parties and application programmers to extend the framework by adding different kinds of media formats, compression schemes, effects processing, and renderers.

In JMF, applications are modeled as a flow graphs — nodes represent media handling modules, while the edges represent media flows. Media data are first read from some input source, such as a file, a network connection, or a capture device. Then the media data are processed in order to perform (de)compression, apply effects, or change the media format. The processing is done by one or more nodes in the flow graph, each represented by, for example, an effect or a codec module. Finally, the processed media data are output to a renderer, a file, or streamed over the network.

Typically, an application programmer specifies the media source and sink by means of URLs, and JMF internally tries to build a flow graph by connecting different components. In other words, the framework takes care of constructing a graph which has the necessary components. As an example, an URL may specify that video data will be read from a multicast address, and the system will configure a graph with the components necessary for receiving the stream from the network, perform decompression, and then display the video on the screen.

Additional control is provided by programmatically interacting with a so-called processor. Some of the components in the graph may be specified, and the rest of the graph construction is then left to the framework. For example, the API provides methods for inserting customized plug-ins, such as codecs and effects, into the audio or video track or for using a customized renderer. The graph building process only takes into account a single host, and there seems to be no descriptions available about how the automatic graph building process works.

4.4 The Dali Multimedia Software Library

Dali [59] is a reusable multimedia software library intended for developing processing intensive multimedia software. Developing high performance multimedia software from scratch is both complex and time consuming, and the development of Dali was motivated by these recognized difficulties. The authors of Dali recognize that applications programmed directly in C may provide high performance, but argue that development is time consuming and the resulting programs complex. Consequently, reuse and maintenance is difficult. On the other hand, they argue that high level libraries are inefficient and provide too little control to the programmers. The abstraction level provided by Dali is described as an intermediate solution, between C programming and using conventional libraries for developing multimedia applications.

The Dali library is designed in such a way that developers maintain control of resource usage. According to this design principle, functions for I/O and memory management are explicitly exposed to the developer instead of making these operations implicit and a side effect of other operations. In the same spirit, Dali programmers are exposed to the structural elements of different media types, such as image

headers and the structural elements of MPEG video sequences. Such headers include sequence headers, group-of-pictures headers, and picture headers. This allows Dali programs to access the different components of JPEG images directly, operate in the compressed domain by processing DCT domain picture data, or access the motion vectors in MPEG video sequences.

In [59] some example programs written in Dali are presented. These rather compact programs illustrate how Dali can be used to implement picture-in-picture effects, MPEG decoding, and de-multiplexing of a MPEG system stream which may contain several audio and video tracks. The authors claim that the performance of Dali applications are competitive with hand tuned C code, and the claims are validated by providing performance measurement comparisons with other software for image compression, image conversion, and MPEG video decoding.

4.5 Infopipe

The Infopipe research project, described in [15], aims at simplifying application development within the domain of media streaming. Middleware platforms for building distributed applications are often based on Remote Procedure Calls (RPC) or Remote Method Invocation (RMI) and are targeted at hiding communication aspects and hence making them transparent. In contrast, the Infopipe approach is to make certain aspects of the communication explicit. The motivation is to support application level monitoring and adaptation in response to varying resource availability. The authors make an analogy between Infopipes for information distribution and plumbing for water distribution.

The Infopipe approach is to define suitable abstractions and hence provide developers with the necessary and sufficient concepts for constructing media streaming applications. The defined building blocks are so-called Infopipe components. An application may instantiate such components and interact by means of operations for monitoring and control. The application may query an Infopipe component about status information, such as the number of video frames that has passed through within a specific time interval.

Different kinds of Infopipe components for communication, control, filtering, and transformation are described in [15]. Supported components include sources, sinks, buffers, filters, pumps, remote pipes, and split and merge tees. Different Infopipe components may be connected in order to establish information flow from sources to sinks, so-called Infopipelines. A connect operation is supported which connects a so-called output port in one Infopipe component to an input port in another Infopipe component. A pump is an active element which drives the data delivery, for example, by pulling information from an upstream component and pushing information

to a downstream component. Remote pipes must be explicitly installed whenever an Infopipeline crosses computer host boundaries. Infopipe components are created at each end of the remote connection and handle communication and synchronization.

The Infopipe approach supports reasoning about Infopipe compositions, both with respect to functionality and QoS characteristics. An example of functional reasoning is to validate if data can actually flow from a source to a sink. An example of reasoning about QoS characteristics is to determine the end-to-end latency for an Infopipeline. The authors of [15] recognize that other QoS properties, such as CPU consumption, do not have strict additive behavior (e.g., due to caching).

The ideas from the researchers in the Infopipe project have influenced the development of GStreamer [77], an open source framework for developing streaming media applications. Hence, the GStreamer framework resembles the Infopipe approach, where different components may be plugged together into pipelines.

4.6 MediaMesh

As described in [9, 10], MediaMesh is an architecture for integrating isochronous processing algorithms into media servers. The purpose of the architecture is to allow different operations on live media streams. Such operations include security (watermarking and encryption), time shifting (fast-forward, pause, and seek), adaptation, and (de)multiplexing of streams. The architecture is also targeted at QoS support. Similar to other media server architectures, the main components are an *application server*, for interacting with users, a *control server*, for admission control, and a *data exporter* which is responsible for moving the media data through the flow graph, from sources to target devices.

Media streams are modelled as directed graphs where the edges represent communication and the nodes represent processing modules. The edges, called pipes, are connected to the processing modules, called filters, via named connection points, called ports. The direction of each edge indicates the direction of the media stream flow, from sources to sinks. Control information on the other hand, is allowed to flow in both directions, that is upstream for requests and downstream for replies. The MediaMesh architecture adds infrastructure in the data exporter for composing graphs and a *graph manager* in the control server for constructing the graphs.

Major features of the MediaMesh architecture include efficient buffer management, distributed stream control, and QoS management. During setup, a pipeline characterization process propagates characteristics from a sink towards the sources. This process allows validation with respect to compatibility between filters. Additionally this setup process allows the filter modules to indicate the required amount of space for headers and trailers, in order to avoid data copying. The ownership of

different control commands is also assigned during the setup phase in order to provide distributed stream control. An example of such a control command is to switch to a different version of a video file in order to change video resolution. Such a control command will flow from the issuing filter to the filter which is responsible for the switch. The graph manager is also informed about the commands which are valid for the pipeline.

The resource management support in MediaMesh is deterministic and considers CPU and memory usage. Each filter module has some associated properties, including CPU and memory requirements. Based on these properties the system may calculate the overall resource requirements for different media flow graphs. This information is then used for admission control, in order to avoid over-utilizing the available resources and hence degrade the performance of already admitted streams. When [10] was written, cross data exporter graphs were not supported. From our understanding this limits the processing of a graph to a single host.

4.7 Parallel Software-only Video Effect Processing

A research effort for exploiting parallel processing of video effects (PSVP) is described in [53–55]. A software-only solution is proposed in order to reduce cost and improve flexibility for the targeted environment, i.e., Internet video. The flexibility is important in order to cope with the dynamic changes in available resources. These changes cause variability in frame rate, delay, and packet loss. Effects, such as fade, blend, and affine transformations are supported as well as the ability to compose video streams. Handling potential complex effects for a number of streams in real-time has driven support for both functional, temporal, and spatial parallelization.

The effect processing is specified in a high level language which is interpreted by a compiler. The compiler generates a directed graph, where the nodes represent video effect operations. The graph is then mapped onto the available processors, each performing a part of the overall effect processing. The authors recognize that partitioning the video data is not straight forward, due to both spatial and temporal dependencies.

Temporal parallelization is realized with a centralized approach where a processor distributes different frames from one or more video streams to different processors [53]. Compared to a decentralized approach, both intra and inter stream synchronization is simplified, although at the cost of increased latency.

For spatial parallelization each processor receives the stream over IP multicast, decodes the full video images, and applies the effects processing on a region. A disadvantage is that each processor receives and decodes video data which are outside the region of interest.

After applying the effects, the different frames and frame regions are combined by a processor into an effect enhanced video stream. In [53] an interleaver is described for the temporal parallelization case, which dynamically adapts buffering time in order to reduce frame dropping. Due to lack of support in standards such as M-JPEG, H.261, H.263, and MPEG for specifying the geometric relationships between the different frame regions, the authors developed a new intermediate semicompressed video format for the spatial parallelization case [54]. The format supports rectilinear subregions, is DCT-based, and allows DCT values to be used directly in the combination phase when generating output stream formats. Small and high frequency coefficients may be dropped when resources are scarce. Additionally, both luminance and chrominance information is encapsulated within the same packets.

A control and coordination scheme for PSVP is described in [55]. The scheme is based on the Scalable Reliable Multicast (SRM) protocol and the Scalable Naming and Announcement Protocol (SNAP). These protocols are based on IP-multicast and provide tunable reliability semantics and recoverable state. The authors argue that IP-multicast provides the required efficiency and also the required transparency with respect to the location and the number of processors which implement an effect.

In [54], performance numbers are presented for a general affine transform effect, executed by one to seven processors. With respect to latency, 250 ms is reported for the one processor configuration. For the seven processor case, temporal parallelization increases the latency to 340 ms, while a reduction to 70 ms is achieved by using spatial parallelization. In the seven processor case, spatial parallelization achieves 50% efficiency, while temporal parallelization reaches 75% efficiency. The authors explain that spatial parallelization efficiency is bounded by the overhead due to receiving and decoding the full video stream before applying the effects.

4.8 Discussion

The targeted application domain for the systems presented in this chapter differ. Some are rather wide in scope, such as Java Media Framework, Dali, and Infopipe. The others are meant for more specific usage — OpenCV for computer vision, MediaMesh for media servers, and PSVP for parallel video effect processing.

With respect to application configuration, the graph building processes in both JMF and MediaMesh do not take more than one computer into account. The configuration of distributed applications is left to the developers. The graph building process in JMF is also automatic and implicit, and we were unable to find any details about the implemented algorithm. In both OpenCV and Dali flow graphs are explicitly programmed by developers, but to the best of our knowledge no support for distributed processing is provided. Infopipe supports explicit connections and allows reasoning

about distributed compositions. Support for distributed processing was a main goal for PSVP. In JMF, Dali, and MediaMesh components are connected directly, without a level of indirection. This makes configuration, reconfiguration, and independent parallelization of the different levels in the video analysis hierarchy difficult. In Infopipe a configuration language has been proposed for explicitly constructing Infopipelines, but a level of indirection is not provided directly. PSVP takes advantage of IP multicast based communication in order to have such a level of indirection and to provide transparency with respect to the number of participants and their location.

Distributed and parallel processing of video streams is not supported in OpenCV, JMF, Dali, or MediaMesh. Some of these systems support multithreading in order to exploit the parallelism inherent in multimedia processing. For Dali, parallel processing by multithreading is referred to as further work. For PSVP on the other hand, the main goal has been distributed and parallel processing. However, the authors of PSVP acknowledge the difficulty of distributing different parts of a video stream to different processors, due to spatial and temporal dependencies within and between video frames. The authors also recognize that sending a full video stream to all processors gives a significant decoding overhead, as confirmed by their reported experiments.

Developing real-time distributed video content analysis applications is challenging. Most of the systems presented in this chapter seem to provide little support for flexible distribution and parallelization of such applications. The approach represented by PSVP is promising, but the expressiveness provided by group-based systems is rather limited. Consequently, it seems difficult to efficiently deliver different parts of the video data to different filters and feature extractors in order to improve efficiency by parallel processing. Altogether, these issues have motivated our research on exploiting distributed content-based event notification services for distributed and parallel video processing.

Chapter 5

Papers and Contributions

An overview of the research contributions of each individual paper included in this thesis is presented in this chapter. Additionally, we provide a discussion of the combined thesis contributions and compare our approaches to the related work presented in Chapter 2, 3, and 4.

5.1 Overview of Research Papers

In the following we describe the main contributions of each paper and how the paper fits with respect to the overall thesis goals. Since each paper is self-contained, some information is necessarily repeated in different papers. For information about where the paper has been published, the evaluation process and outcome, and the contributions made by the different authors, the reader is referred to the cover pages of each research paper, presented in Part II. In short, Eide has been the driving force behind the work presented in Paper I, II, IV, VI, and VII, while Paper III, V, and VIII have been written cooperatively within the project. For the project papers, the following description will focus on the contributions made by Eide.

The slides and posters used for presenting the papers at conferences and workshops are also available from the project web pages [3].

Paper I: Supporting Distributed Processing of Time-based Media Streams

Some general aspects of distributed processing of time-based media streams are addressed in this paper. The paper motivates the need for supporting real-time processing of time-based media streams and suggests a general way of structuring applications within this domain. An architecture is presented, and the paper investigates de-

sirable system support for building such applications. In particular the paper focuses on issues related to the interaction model as well as time and synchronization.

The proposed framework assumes a notion of global time and supports specifications of temporal relationships by associating a time interval to each event. The framework does not itself address the issue of time synchronization, but suggests that approximation to global time may be realized by synchronizing computers by, for example, the Network Time Protocol (NTP) (RFC 1305).

Identified desired framework characteristics include reuse-ability, scalability, performance, resource management, and fault tolerance. The paper argues that modularization, distribution, adaptation, reconfiguration, migration, and replication are important mechanisms in order to provide such characteristics. Based on the framework architecture, the desired characteristics, and the communication requirements, the paper suggests an event-based interaction model. The paper argues in favor of event-based communication, due to the level of indirection provided and thereby the loose coupling achieved between components.

The paper also describes a prototype which takes advantage of Mbus, a communication system which resembles event-based communication. The Mbus addressing scheme allows a message to be addressed to number of receivers, by means of attribute names and values, as described in Section 2.5.

The main contribution of this paper from a thesis perspective is the foundation that it provides for the rest of the thesis work. The computational architecture is introduced, requirements derived, and the design issues are discussed and evaluated. The paper describes a first prototype implementation, which allowed testing and provided useful feedback on the architecture.

Paper II: Real-time Processing of Media Streams: A Case for Event-based Interaction

In this paper, the communication requirements for the targeted application domain are analyzed in more depth, compared to Paper I.

The paper argues that the communication patterns and requirements fit well with the publish/subscribe interaction paradigm and hence event-based communication. The application requirements are then translated into requirements for the event notification service. The data model for notifications as well as event notification selection, filtering, delivery, and ordering issues are discussed in detail, and we argue that a suitable event notification service may satisfy the corresponding requirements.

The paper also advocates the potential unleashed by an event notification service capable of streaming video data. Such a service will allow a video coding scheme to be developed, where different video receivers may express interest in only certain

parts of the video signal, both temporally and spatially. This will allow both network and processing resource consumption to be reduced, and subsequently efficiency to be improved.

By analyzing the communication requirements in more detail, requirements for a suitable event notification service are identified. Thereby the paper lays the foundation for exploring event-based communication further for the targeted application domain, and in particular for video streaming.

Paper III: Scalable Independent Multi-level Distribution in Multimedia Content Analysis

In this project paper, a framework is proposed where each level of a content analysis task can be parallelized and distributed independently. As an example, one particular feature extraction task can be split into a number of sub tasks, each executing on different computers. Due to event-based and indirect communication, the analysis tasks at higher or lower levels do not need to be modified when a particular level is parallelized, neither programming nor configuration wise. The components involved in the classification part of the application subscribe to the information of interest, and whether this information has been produced by a single or a number of feature extraction components does not matter. The event notification service is responsible for delivering the relevant information to the respective interested parties. If for example the feature extraction part of the application experiences a bottleneck situation, it may be parallelized and executed by a number of computers. The processing resources can be focused at the bottleneck at hand, simplifying both development and deployment.

In order to validate the framework, a prototype application was implemented for real-time detection and tracking of a moving object within a video stream. The paper presents experimental results for different configurations of this real-time motion vector based object tracking application, executed by one to ten processors. Even for this relatively tightly coupled application, where different components interact with the video frame rate frequency, reasonable scalability is achieved. The paper argues that a massively distributed application, utilizing a large number of cameras, may require such tight coupling only between some components. However, the measurements reveal the need for a more fine grained video streaming solution, compared to standard IP multicast based streaming. The scalability of the feature extraction part suffered, as each feature extraction component had to both receive and decode the full video stream, before processing only a region within each frame.

The results demonstrate that our framework allows construction of scalable applications by means of distribution and parallelization of the different logical application levels, i.e., streaming, filtering/transformation, feature extraction, and classification.

Paper IV: Extending Content-based Publish/Subscribe Systems with Multicast Support

An approach for extending distributed content-based publish/subscribe systems with multicast support is proposed in this paper. This is a well known challenge in the research community and is motivated by the potential efficiency and performance gains, as described in [18, 41].

Our earlier prototypes used Mbus for communication, in addition to IP multicast based video streaming. However, the end node filtering approach of Mbus is not appropriate as the volume of data transported through the service increases, among others as a result of transporting several video streams through the service concurrently. Distributed content-based event notification services on the other hand, filter notifications close to the source, provide clients with more expressiveness for subscribing, and give more flexibility with respect to the choice of transport mechanisms.

The paper presents an architecture for a distributed content-based event notification service, capable of exploiting native (network level and link level) multicast support. The service is intended for intradomain and LAN usage. As stated in the paper, we view our approach as complementary to approaches for WAN usage, as we envision that intradomain services connect to a WAN event notification service through gateways. Such a division between intradomain and interdomain protocols have also been successful elsewhere, for example for IP routing.

Central to the suggested approach is a so-called mapping specification, which maps all event notifications potentially generated to a number of different communication channels, for example IP multicast addresses. The paper does not address algorithms for automatically calculating mapping specifications, but suggests an approach for manually determining such specifications. In other words, the mapping problem is separated out and the other required mechanisms are implemented. Such mapping specifications can be generated off-line, online, manually, or automatically and may be changed during runtime. Thereby, a new mapping specification can be installed, which is more appropriate in relation to the currently generated notifications and subscriptions, allowing the service to adapt to the current application generated load.

When a client publishes notifications through a server, the server maps each notification to an IP multicast address. The notification is thereby efficiently forwarded to all other servers responsible for clients which have matching subscriptions. This inter server communication is transparent to clients.

The presented experimental results indicate the potential for the service to provide both scalability and high performance. The measured performance shows that a client may publish several thousand event notifications, carrying several MBytes of data, per second. Due to the use of native multicast, the service is unaffected by the number of

clients having interest in the same notifications. Additionally, the distributed architecture of the service provides scalability, since computers using the service contribute by hosting part of the distributed service.

From a thesis point of view, this paper demonstrates that a distributed content-based publish/subscribe system may provide high performance. In particular the results show that even the video streaming part of an application can be handled by a content-based event notification service.

Paper V: Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis

This paper was written collaboratively in the project and builds on earlier results. The paper shows how requirements for timeliness, i.e., latency and temporal resolution, as well as accuracy can be handled by a scalable and resource aware architecture for the domain of real-time distributed video content analysis.

In the suggested approach applications are represented as graphs, where the nodes represent processing tasks and the edges represent the directed flow of data. A QoS model for real-time content-based video analysis is also presented. Given such an application graph, a specification of timeliness and accuracy requirements, and a model of the physical processing environment, the paper describes QoS aware mapping of the application onto the distributed processing environment.

In the proposed architecture a distributed content-based event notification service is responsible for all inter component communication. The paper advocates that event-based interaction simplifies (re)configuration and additionally supports independent parallelization at different logical levels.

With respect to scalability the content-based event notification service plays a central role, and the paper describes how scalability can be achieved for video streaming, filtering/transformation, feature extraction, and classification. The paper argues that by streaming video over a content-based event notification service, receivers are provided with the ability to customize the video stream.

When the experiments reported in this paper were conducted, the implementation of this video coding scheme was not ready. The empirical results which demonstrate the scalability of the object tracking application are from Paper III, while the performance measurements of the event notification service are from Paper IV. The presented numbers demonstrate the ability of our service to handle the data rates required for video streaming, and the distributed architecture of the service also allows concurrent streaming of several video streams.

Paper VI: Exploiting Content-Based Networking for Video Streaming

This technical demonstration paper gives an overview of how content-based networking may be exploited for heterogeneous multi-receiver video streaming.

The novelty of the presented approach is that each video receiver is provided with independent and fine grained selectivity along several different video quality dimensions, while efficiency is maintained on the sender side, in the network, and on the receiver side. Consequently, each video receiver may independently customize the video stream according to user preferences and available resources, such as network bandwidth, CPU performance, power availability, and display capabilities. The video coding scheme supports customization of the video signal with respect to region of interest, signal to noise ratio, colors, and temporal resolution.

From a thesis perspective this paper lays the foundation for more efficient distributed and parallel video content analysis. The potential efficiency improvement is due to the ability for each video receiver to express interest in only a subset of the full video signal, spatially and/or temporally.

Paper VII: Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming

Compared to Paper VI, this paper describes in more detail how the field of content-based networking can be bridged with well known techniques from the fields of video compression and streaming in order to support fine granularity multi-receiver video streaming. The novelty is that each video receiver is provided with fine grained selectivity and therefore may customize the video stream in order to trade user preferences against available resources, such as network bandwidth, computational resources, power availability, and display resolution. The supported dimensions in our video coding scheme are region of interest, signal to noise ratio, colors, and temporal resolution.

The paper provides details on how fine grained selectivity can be realized in the different video quality dimensions. Performance measurements for a prototype implementation are also presented, which show that both bandwidth and processing requirements drop accordingly when video quality is reduced in the different dimensions. From a real-time video content-analysis point of view these results are encouraging, because they indicate that different components executing on different computers in a content analysis application may subscribe to and hence receive only a certain part of the video signal. In other words, each component is provided with full flexibility along the different video quality dimensions. The customization al-

lows efficiency to be improved, since only the required part of the video signal is received and decoded. Consequently, the consumption of network and computational resources is reduced, creating a potential for improving scalability.

These results are also promising outside the domain of distributed and parallel real-time video content analysis. The paper contributes to state of the art by demonstrating how video streaming over content-based networking is able to support scalable and more fine-grained multi-receiver video streaming, compared to other techniques which often use unicast or multicast directly.

Paper VIII: Real-time Video Content Analysis: QoS-Aware Application Composition and Parallel Processing

This project paper extends the work presented in Paper V. For communication, a distributed content-based publish/subscribe is now used throughout the whole architecture. For performance reasons, the paper advocates a publish/subscribe system capable of exploiting native multicast support, as described in Paper IV. The suggested approach for streaming of video is the fine granularity multi-receiver video streaming system presented in Paper VI and VII. A new prototype application was developed for validation purposes. Measurement results are presented for the updated version of the object tracking application, where all communication is handled via our IP multicast enabled distributed content-based publish/subscribe service, including the video streaming. Compared to earlier published results, the measurements indicate significantly improved scalability for spatial parallelization of video processing. In case of video content analysis, this represents the filtering, transformation, and feature extraction parts of the applications.

It should be noted that the feature extraction task used for the experiments (i.e., motion vector calculation) is somewhat challenging, due to spatial and temporal dependencies. In order to calculate a motion vector for a single 16×16 pixel block, pixel data from neighboring blocks are required. Therefore, each motion vector calculation component must subscribe to a region somewhat larger than the regions for which it calculates motion vectors. Consequently, when motion vector calculation is performed in parallel by a number of components, some video blocks are received and decoded by several components. This introduces some overhead which is related to this specific feature extraction task, but not inherent in the framework itself.

In spite of the spatial dependencies, the presented results indicate that the overhead is quite low. Supported by the measurements presented in Paper VII, the overhead should be close to zero for other kinds of feature extraction algorithms, where different components subscribe to strictly isolated frame regions. Additionally, the motion vector components only subscribe to the gray level part of the video signal,

thereby eliminating the need for receiving and decoding the color part of the video signal. These results represent a substantial improvement compared to our and other earlier published results, as reported in [33, 54].

From a thesis point of view, this most recent paper demonstrates that event-based communication provides some distinguishing advantages for the application domain of real-time distributed and parallel video content analysis. The results obtained by integrating (1) a distributed IP multicast capable content-based event notification service and (2) a fine granularity multi-receiver video streaming scheme which exploits such a service into (3) a real-time distributed and parallel video content analysis application, allow us to confidentially claim validity of our approach.

5.2 Discussion

In this section we provide a discussion of the contributions of this thesis work in relation to the related work presented in Chapter 2, 3, and 4 and the open issues discussed at the end of each of these chapters. The discussion is structured into the same three subsections, i.e., many-to-many communication, multi-receiver video streaming, and video processing.

5.2.1 Many-to-Many Communication

The motivations for us to exploit content-based publish/subscribe systems for real-time video content analysis were the fine granularity selectivity and the improved flexibility by having an additional level of indirection. The additional level of indirection allows event-based systems to take advantage of different underlying transport technologies. How the service is realized can be addressed without changing anything in the applications, and concerns regarding configuration, reconfiguration, and deployment can be handled more separately. The mapping onto underlying communication can also be changed during runtime in order to better fit the current flow of notifications.

We have also argued that it is reasonable to handle the local area/intradomain case differently from the wide area case, represented by for example interdomain or Internet usage. We envision that intradomain and interdomain protocols will be connected hierarchically. Hierarchical protocols are often considered beneficial elsewhere too. For example in IP routing, hierarchical protocols allow policies to be specified differently for the intradomain and the interdomain cases. Based on this reasoning, we view our work on an efficient local area/intradomain content-based publish/subscribe service as complementary to services for the wide area/interdomain case.

To the best of our knowledge, existing content-based publish/subscribe systems did not exploit native multicast support. Therefore, we extended an existing distributed content-based publish/subscribe system with IP multicast support in order to provide efficient dissemination of notifications. This thesis has not addressed the issue of developing algorithms for automatically calculating mapping specifications, i.e., determining a mapping from the content-based event notification space to multicast addresses. So far we have implemented the mechanisms which have allowed us to experiment with manually generated mapping specifications and thereby demonstrate the potential of our approach.

Altogether, the added complexity, compared to using group-based communication directly, seems manageable. The experimental results presented in Paper VIII demonstrate the benefits and potential of our approach. The experiences gained so far have led us to claim that high-performance content-based publish/subscribe systems are better suited for real-time video content analysis than using group-based communication directly.

5.2.2 Multi-Receiver Video Streaming

Our early prototypes were based on video streaming over IP multicast. Experiences with these prototypes and the lack of solutions for fine granularity multi-receiver video streaming motivated our research on exploiting content-based networking for video streaming.

With the exception of the systems specifically made for IP multicast based video streaming, all systems described in Chapter 3 rely on some kind of overlay network infrastructure. Our approach is no different in this respect, as content-based event notification services are also realized as overlay networks.

With respect to adaptation, the number of supported video quality dimensions in our approach and the granularity of each dimension give rise to a large adaptation tradeoff space. Although we have some ongoing activity within this area in the project, the work described in this thesis has not addressed the issue of adaptive video streaming.

In our video streaming approach additional processors for partitioning the video data are not needed, since this is handled by the video coding scheme and the content-based publish/subscribe system. The video signal is partitioned with fine granularity at the sender side, and the need for additional application level filtering and transformation in the end systems is thereby largely reduced.

Consequently, in our approach each video receiver may independently and arbitrarily customize the stream along the different video quality dimensions. In effect CPU and bandwidth resource consumption are reduced accordingly.

5.2.3 Video Processing

The video processing part of our early prototypes were based on the Java Media Framework. However, some of the other systems described in Chapter 4 could have been used instead. For our later prototypes the benefits of using these already existing systems seem less obvious. As an example OpenCV includes different video processing functions which could have been used as building blocks, but as stated in Section 1.5, this thesis has not addressed the development of new kinds of filters or feature extractors. Similarly we could also have used for example Dali for video processing, but that would have required integration work with respect to our video coding scheme.

Our framework allows video content analysis applications to be decomposed functionally into streaming, filtering, feature extraction, and classification tasks, which can be then be executed in a distributed fashion. Similarly to PSVP, which relies on IP multicast, our approach is also based on a communication system which provides a level of indirection, and thereby provides transparency with respect to the number of participants and their location. The glue is the distributed content-based event notification service. Each component may subscribe to the event notifications of interest, and by whom and where it has been generated does not matter. This simplifies both configuration and reconfiguration and allows different components to be deployed within the address space of a single process, within different processes on a single computer, or hosted by different computers. Thereby, the configuration, reconfiguration, and deployment concerns can be handled more separately[1]. The ability to map event-based communication onto different underlying communication technologies and the opportunity for mapping different parts of the event notification space to different communication channels also allow communication service performance issues to be handled separately.

In addition to support for distributed processing and functional parallelism, our framework supports spatial parallelization of filtering, transformation, feature extraction, and classification tasks. The level of indirection provided by content-based event notification services also simplifies parallelization, since each level of the content analysis hierarchy may then be independently parallelized. The most complete description and evaluation is provided in Paper VIII. By spatial parallelization latency can be reduced, which is important for interactive and feedback control applications.

With respect to video processing efficiency, performance numbers are reported in Paper VIII for an application which utilizes functional and spatial parallelization. The reported overhead for performing feature extraction in parallel on a number of computers is small and most likely due to the dependencies inherent in motion vector

[1]The development of algorithms for determining application configurations has in the project been addressed by Granmo [45].

estimation. The overhead seems to be caused by the fact that different motion estimation components receive and decode some of the same video data. In other words, the overhead is related to a particular algorithm and not the framework itself. Furthermore, the measurements presented in Paper VII indicate that the overhead will be more or less zero for other kinds of feature extraction algorithms, which process strictly disjunct parts of the video data.

By combining our IP multicast capable distributed content-based publish/subscribe system, a fine granularity multi-receiver video streaming scheme, techniques for functional and spatial parallel processing of video data, and state of the art techniques for distributed classification, we have successfully validated our approach. Consequently, this work has demonstrated the strength of exploiting event-based communication for real-time distributed and parallel video content analysis.

Chapter 6

Conclusion and Further Work

This chapter first revisits the research topics and goals of this thesis. Then a description of the major contributions is given, before some critical remarks are presented and discussed. Lastly, some opportunities and ideas for further research are provided.

6.1 Research Topics and Goals

The motivation for the work presented in this thesis has been to provide support for the application domain of real-time distributed and parallel video content analysis. The challenges represented by analyzing massive amounts of video data in real-time have spurred the research on suitable techniques for communication, streaming, filtering, and feature extraction. The research has evolved around exploiting event-based communication as the main communication mechanism for this challenging application domain. In particular, the following goals have been identified:

- Investigate if event-based interaction is a good fit for the application domain of real-time distributed and parallel video content analysis
- Investigate if event-based communication is suitable for streaming real-time video data in particular and transporting high data rates in general
- Investigate if event-based communication can support flexible distribution and parallelization as well as efficient execution of such applications

6.2 Major Contributions

The contributions presented in this thesis have been published in a number of research papers [32–39]. The contributions are within three areas — event-based communi-

cation, video streaming, and real-time distributed and parallel video processing. The context of the research, real-time distributed video content analysis, connects these areas tightly. However, this does not mean that the usefulness of the results are limited to this particular domain. On the contrary, we claim that some of these results are useful in general, and not limited by the scope of the DMJ project.

Additionally, the software for the content-based publish/subscribe system and video streaming has been made available as open source from the DMJ project web pages [3]. The intention is to allow others to validate our results, for example by repeating some of the experiments. By allowing others to modify the software, the opportunities for building on our work in future research are also significantly improved.

In the following subsections, the thesis contributions within the areas of event-based communication, video streaming, and real-time distributed and parallel video processing are summarized.

6.2.1 Event-Based Communication

This thesis demonstrates that event-based interaction is well suited for the domain of real-time distributed video content analysis. In [34, 35], arguments which support this claim are given. Event-based systems differ with respect to the data model for the notifications and the expressiveness of the subscription language. Content-based systems offer most expressiveness and hence flexibility.

A distributed content-based overlay network have similarities with IP multicast. Pruning of messages is done upstream, while replication is done downstream. Compared to group-based systems, which basically allow clients to join and then receive messages destined to the group, content-based systems provide much more fine-grained selectivity. The price for the additional flexibility is complexity. Despite the added complexity, this thesis shows that distributed content-based event notification services are both suitable and beneficial for real-time distributed video analysis.

For handling the massive amounts of data and the real-time requirements, we have extended an existing distributed content-based publish/subscribe system with IP multicast support, as described in [36]. To the best of our knowledge, IP multicast support was not implemented in any other content-based publish/subscribe systems at that time. By mapping content-based publish/subscribe onto IP multicast, efficient dissemination of notifications is achieved. Performance numbers presented in [36] show that each client may publish several thousand notifications per second, carrying several MBytes of data per second. This is more than sufficient for streaming high quality video. This system was also used experimentally for a real-time video content analysis application, as described in [39]. All communication, even the video streaming, was handled by the content-based publish/subscribe system.

Hence, the experiences gained so far have led us to claim that high-performance content-based publish/subscribe systems are well suited for the domain of real-time distributed and parallel video content analysis. This thesis has also demonstrated that content-based publish/subscribe systems offer significant advantages compared to other alternatives, including systems which use group-based communication directly.

Additionally, event-based interaction is recognized as being well suited for loosely coupled distributed applications in general. Efficient and high performance event notification services will allow content-based publish/subscribe to be used in other application areas as well, for example within the fields of sensor networks and high performance computing.

6.2.2 Fine Granularity Multi-Receiver Video Streaming

This thesis also demonstrates how content-based event notification services can be exploited for fine granularity multi-receiver video streaming. A prototype has been developed, and the video coding scheme as well as performance numbers are presented in [37, 38]. The thesis contribution in this area is the bridging of techniques from the fields of video compression and streaming with content-based networking. Our video coding scheme has been specifically developed to exploit the powerful routing capabilities of content-based networks.

In our approach, video receivers are provided with fine granularity selectivity along different video quality dimensions and they may independently customize the video signal with respect to region of interest, signal to noise ratio, colors, and temporal resolution. Efficient delivery, in terms of network utilization and end node processing requirements is maintained, as demonstrated experimentally in [38, 39].

Such fine grained selectivity is required in order to maintain efficiency within the domain of real-time distributed and parallel video content analysis, due to the fact that different computers may process different parts of a video stream, functionally, spatially, and temporally.

Additionally, a video streaming solution for handling heterogeneity in a scalable manner is also useful outside the video content analysis domain. Our scheme represents an efficient way of streaming video data to a number of receivers, in spite of differences in network availability, end node capabilities, and receiver preferences.

Consequently, this work contributes to state of the art by demonstrating how video streaming over content-based networking is able to support heterogeneous multi-receiver video streaming. In our view, this represents a promising approach in the domain of multi-receiver video streaming for efficiently handling the huge and increasing diversity in device capabilities and resource availability, amplified by the rapid progress in wireless, mobile, small scale, and ubiquitous computing.

6.2.3 Real-Time Distributed and Parallel Video Processing

Video streaming over content-based networking reduces the need for application level filtering and transformation of video data. Additionally, for distributed and parallel processing, the efficiency is improved compared to other existing alternatives. The reason is that the match between what is needed by different computers and what is delivered and decoded by each computer can be improved. In other words, the amount of redundant calculations is reduced. This was demonstrated experimentally, as reported in [39].

The event-based interaction also provides a level of indirection — a key factor for flexible and independent distribution and parallelization of each logical level. In effect, the available processing resources can be focused on the processing bottlenecks at hand. Additionally, the level of indirection also allows different transport mechanisms to be used for event notification dissemination. IP multicast has been used in the experiments reported so far, but other technologies could have been used. Examples include other group-based systems and/or unicast technologies. Consequently, application development is more decoupled from quality of service mapping and deployment, as described in [32, 33, 39].

In combination, distributed high-performance content-based publish/subscribe systems, video coding schemes which exploits the powerful routing capabilities of such systems, and distributed and parallel video processing provide a promising foundation for developing efficient real-time video content-analysis applications. This was demonstrated by integrating the results presented in this thesis with the classification work done by Granmo [45], as described in [39]. The measurements reported in [39] reveals the scalability of a specific application, by demonstrating that the overhead of distributed and parallel processing can be kept low. However, it seems reasonable that the results also indicate the scalability of our approach in general. Consequently, as the overhead can be kept low, our approach allows improved quality of service (QoS) requirements and thereby harder performance requirements to be satisfied by adding computational resources, such as computers and network capacity.

6.3 Critical Remarks

Having summarized the main results both with respect to their usefulness in the context of the targeted application domain and in general, this section presents some critical remarks. As stated in Section 1.4, the research method that has been used in this thesis work is the design paradigm. An inherent characteristic of the research method associated with the design paradigm is that results can only be validated, and not proven in the mathematical sense.

We have followed the design paradigm and performed requirement analysis, generated specifications based on these requirements, designed the systems, and implemented these designs. The developed prototypes have been tested and formed the bases for experiments. Experiments have been conducted for the individual areas addressed in this thesis, i.e., event-based communication and fine granularity multi-receiver video streaming. The techniques from these individual areas have also been combined with state of the art techniques for distributed classification, and integrated into a real-time distributed and parallel video content analysis application for experimental purposes. The results obtained by integration and the reported experimental measurements represent significant improvements when compared to relevant work. However, we recognize that for example simulations could have been used for more large scale experiments.

One may argue that other or more challenging applications should have been developed, which more completely span the targeted application domain. Clearly, this may have supported our claims more firmly. Closely related is the issue of validating scalability, which leads to questions about the scale of the experiments. One may argue that the conducted experiments are too limited in different dimensions, for example with respect to the maximum number of computers used and the number of video streams analyzed concurrently. Similarly, the IP multicast enabled distributed content-based publish/subscribe system could have been evaluated more closely, by larger scale experiments or by using simulations. In this respect it should be noted that our work has been targeted at intradomain and LAN usage, i.e., our work is complementary to event notification services for WANs where scalability challenges are much harder. For research within the design paradigm, which rely on prototyping, the issue of scalability can most likely always be questioned. Additionally, our claims are not solely based on experiments, but also on arguments which have been presented to the research community in the form of peer reviewed papers and presentations at conferences and workshops. By making the software available as open source, we have also made validation by other researchers practically possible.

With respect to the prototypes and experiments, it should be noted that the focus has not been to achieve the best possible absolute performance for a particular application implementation. Rather, in order to determine what is reasonable to expect for applications within the targeted domain we have concentrated on observing the relative numbers, such as the trends for efficiency and latency, as computers are added for distributed and parallel processing.

The focus of our work on real-time distributed video content analysis applications has been on the data flow part and not on the signaling and control parts. The publish/subscribe interaction paradigm may not be suitable for the control and management parts, which may require explicit addressing of endpoints in addition to authen-

ticated, encrypted, and reliable communication. For such purposes, direct one-to-one communication running on top of for example TCP may prove more well suited.

Altogether, we think that our claims have been sufficiently justified. Thereby, they provide adequate ground for claiming validity of our approach, although we also recognize that there are a number of interesting issues raised by this thesis work. In the following section we present and discuss some of these issues, as opportunities for further research.

6.4 Further Work

There are many possible directions for further exploring some of the individual results presented in this thesis. The combination of these results also presents further research opportunities within the application domain of real-time distributed multimedia content analysis. In the following, some ideas for further work within these different areas are discussed.

6.4.1 Event-Based Communication

With respect to event-based communication it would be interesting to leverage on advances within the field of distributed content-based publish/subscribe systems. During the last couple of years, researchers have started to investigate peer-to-peer systems for dynamically constructing and maintaining such content-based overlay networks [64, 78]. An interesting approach is to exploit network level multicast in such systems, for efficient communication between peers located within the same area or domain. In our view, it seems reasonable to handle the LAN/intradomain case differently from the wide area case (e.g., the interdomain/Internet case). In order to take advantage of native multicast support, some kind of mapping approach is required, which maps from the space of potential generated event notifications to multicast addresses. This would require algorithms for automatically calculating such mapping specifications, as described in [46, 61]. Input to such algorithms, will take into consideration information about the current flow of notifications, i.e., the current "traffic" pattern. Such functionality seems reasonable to include as a part of the automatic construction and maintenance tasks for such overlay networks. We plan to look more into this problem in our future work. Research in this direction would also fit quite well to the vision of autonomous computing, a field which has recently gained much attention and where important system characteristics include self-monitoring, self-healing, self-adaptive, and self-reconfiguring behavior.

Quality of service issues for event-based communication are also interesting in cases where different parts of applications have different requirements with respect

to for example throughput, reliability, and delay. Clients may indicate such quality of service parameters in subscriptions and advertisements and the service may use this information to select between different underlying transport protocols. A differentiation between the local area case and the wide area case may also prove useful for quality of service issues. As an example, the combination of reliability and multicast may be achieved by simpler and more efficient protocols, which may even take into account characteristics of the underlying layer two network.

6.4.2 Multimedia Streaming

Today, peer-to-peer based streaming systems attract a lot of research activity. For multi-receiver streaming, end-system multicast represents a promising direction. However, this thesis demonstrates the additional flexibility achieved by exploiting content-based networking for video streaming, compared to direct use of multicast and group-based solutions. Hence, dynamically constructed and maintained content-based overlay networks, based on peer-to-peer technologies, seem attractive for fine grained multi-receiver real-time video streaming.

Another interesting area to investigate is a video coding scheme for content-based networking which is more closely integrated with state of the art video coding techniques, represented by for example H.264/AVC [75]. In H.264/AVC macroblocks are organized into self-contained slices and the so-called flexible macroblock ordering (FMO) technique allows much freedom when organizing macroblocks into such slices [75]. It would also be interesting to look at the potential represented by state of the art scalable video coding techniques [58], such as motion-compensated spatiotemporal wavelet coding in combination with content-based networking.

In order to improve robustness and resilience to transient network failures and congestion, error correcting coding schemes seem like a viable path. This may provide video receivers with the ability to customize, by means of subscriptions, the tradeoff between the amount of redundant information received and the robustness. Interesting in this respect are variants of Multiple Description Coding [82] which are based on forward error correction techniques.

For parallel processing, our video coding scheme has so far been used for functional and spatial parallelization. Support for temporal parallelization can be realized by adding a single attribute/value pair in notifications carrying video data. This attribute will indicate a sequence number for a group of pictures and allow different components to subscribe to these different groups of pictures. Each video receiver may then arbitrarily combine this new dimension with the other dimensions, allowing for example a combination of spatial and temporal parallelization. The added attribute will represent an independent video quality dimension, which further illustrates the flexibility of utilizing content-based networking for video streaming.

Based on experiences gained in the area of real-time video streaming, it seems reasonable to investigate the opportunities for using content-based networking to stream other media types as well, such as audio data. In spite of bandwidth requirements for audio being an order of magnitude less demanding than for video, there are apparently exploitable similarities. Such a scheme may allow each audio receiver to independently customize the audio stream along quality dimensions such as sample size, sampling frequency, number of audio channels, and maybe also resilience.

6.4.3 Real-Time Distributed Multimedia Content Analysis

In order to further validate the proposed architecture, we would like to scale up prototype applications and experiments along several different dimensions, as discussed in the following.

In future experiments it would be interesting to analyze several video streams concurrently. This would allow information from different video streams to be related in both space and time during classification (e.g., for tracking objects both spatially and temporally). So far, our research efforts have been concentrated on video analysis. Further work should consider using different media types concurrently. Features extracted from different media types may then also be related, improving robustness even further [81]. Some applications may improve accuracy by analyzing media data of higher quality. For video, this may translate into higher spatial and/or temporal resolution, allowing more fine grained details to be detected. Additionally, by increasing the frame rate, the responsiveness of feedback control systems may be improved. For some applications, improved temporal resolution is important, even at the cost of increased latency. Hence it would be useful to include temporal parallelization in further experiments, maybe also in combination with spatial parallelization. In case of video analysis, each feature extractor component may then subscribe to and hence process only some of the frames or only a region within some of the frames.

Increases along these different dimensions would have to be met by additional computational resources. In this respect, example applications which geographically span long distances may provide useful feedback on the architecture. It would also be interesting to host applications, or parts thereof, on clusters and in grid like environments. Grid systems have not yet been designed to handle real-time applications, as described in [44]. Hence, real-time distributed and parallel multimedia content analysis represents a challenging application domain which may influence research in grid technologies.

Bibliography

[1] ACM Digital Library. http://www.acm.org/dl/.

[2] CiteSeer.IST. http://citeseer.ist.psu.edu/.

[3] The Distributed Media Journaling project. http://www.ifi.uio.no/~dmj/.

[4] Google. http://www.google.com/.

[5] IEEE Xplore. http://ieeexplore.ieee.org/.

[6] SpringerLink. http://www.springerlink.com/.

[7] The Digital Bibliography & Library Project. http://dblp.uni-trier.de/.

[8] Kevin C. Almeroth. The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment. *IEEE Network, Special Issue on Multicasting*, 14(1):10–20, January/February 2000.

[9] Lisa Amini, Jorge Lepre, and Martin G. Kienzle. Distributed stream control for self-managing media processing graphs. In John Buford and Scott Stevens, editors, *Proceedings of the seventh ACM International Conference on Multimedia, October 30 - November 05, 1999, Orlando, Florida, USA*, volume 2, pages 99–102. ACM, 1999.

[10] Lisa Amini, Jorge Lepre, and Martin G. Kienzle. Mediamesh: An architecture for integrating isochronous processing algorithms into media servers. In Klara Nahrstedt and Wu-chi Feng, editors, *Multimedia Computing and Networking (MMCN'00)*, volume 3969, pages 14–25. SPIE, 2000.

[11] Phillip G. Armour. The Five Orders of Ignorance. *Communications of the ACM*, 43(10):17–20, October 2000.

[12] David Beymer, Philip McLauchlan, Benn Coifman, and Jitendra Malik. A Real-time Computer Vision System for Measuring Traffic Parameters. In *Computer Vision and Pattern Recognition (CVPR'97), San Juan, Puerto Rico*, pages 495–501. IEEE, June 1997.

[13] Ken Birman, Robert Constable, Mark Hayden, Jason Hickey, Christoph Kreitz, Robbert van Renesse, Ohad Rodeh, and Werner Vogels. The Horus and Ensemble Projects: Accomplishments and Limitations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, IEEE Computer Society Press, pages 149–160, January 2000.

[14] Kenneth P. Birman, Robbert van Renesse, and Werner Vogels. Spinglass: Secure and Scalable Communications Tools for Mission-Critical Computing. In *DARPA Information Survivability Conference and Exposition II (DISCEX '01)*, volume 2, pages 85–99. IEEE, 2001.

[15] Andrew P. Black, Jie Huang, Rainer Koster, Jonathan Walpole, and Calton Pu. Infopipes: An abstraction for multimedia streaming. *Multimedia Systems*, 8(5):406–419, 2002.

[16] Michael Blome and Mike Wasson. DirectShow: Core Media Technology in Windows XP Empowers You to Create Custom Audio/Video Processing Components. *Microsoft, MSDN Magazine*, 17(7), July 2002.

[17] Gary Bradski, Adrian Kaehler, and Vadim Pisarevsky. Learning-Based Computer Vision with Intel's Open Source Computer Vision Library. *Intel Technology Journal*, 9(2):118–130, 2005.

[18] Antonio Carzaniga, David S. Rosenblum, and Alexander L Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[19] Antonio Carzaniga, Matthew J. Rutherford, and Alexander L. Wolf. A Routing Scheme for Content-Based Networking. In *Proceedings of IEEE INFOCOM, Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 918–928, Hong Kong, China, March 2004.

[20] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8):1489–1499, October 2002.

[21] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays. In *Proceedings of IEEE INFOCOM, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1510–1520, 2003.

[22] Desmond Chambers, Gerard Lyons, and Jim Duggan. Stream Enhancements for the CORBA Event Service. In *Proceedings of the ACM Multimedia (SIGMM) Conference, Ottawa*, pages 61–69, October 2001.

[23] Desmond Chambers, Gerard Lyons, and Jim Duggan. A Multimedia Enhanced Distributed Object Event Service. *IEEE MultiMedia*, 9(3):56–71, 2002.

[24] Chao Chen, Zhanfeng Jia, and Pravin Varaiya. Causes and Cures of Highway Congestion. *Control Systems Magazine, IEEE*, 21(6):26–32, December 2001.

[25] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8):1456–1471, October 2002.

[26] Jon Crowcroft and Markus Hofmann, editors. *Networked Group Communication, Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001, Proceedings*, volume 2233 of *Lecture Notes in Computer Science*. Springer, 2001.

[27] Jon Crowcroft and Ian Pratt. Peer to peer: Peering into the future. In Enrico Gregori, Giuseppe Anastasi, and Stefano Basagni, editors, *NETWORKING Tutorials*, volume 2497 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2002.

[28] David E. Culler, Deborah Estrin, and Mani B. Srivastava. Guest editors' introduction: Overview of sensor networks. *IEEE Computer*, 37(8):41–49, 2004.

[29] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Trans. Comput. Syst.*, 8(2):85–110, 1990.

[30] Peter J. Denning. Computer Science: The Discipline. In Anthony Ralston and David Hemmendinger, editors, *2000 Edition of Encyclopedia of Computer Science*. 2000.

[31] Peter J. Denning, Douglas Comer, David Gries, Michael C. Mulder, Allen B. Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Communications of the ACM (CACM)*, 32(1):9–23, 1989.

[32] Viktor S. Wold Eide, Frank Eliassen, Ole-Christoffer Granmo, and Olav Lysne. Scalable Independent Multi-level Distribution in Multimedia Content Analysis. In *Proceedings of the Joint International Workshop on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS), Coimbra, Portugal*, LNCS 2515, pages 37–48. Springer-Verlag, November 2002.

[33] Viktor S. Wold Eide, Frank Eliassen, Ole-Christoffer Granmo, and Olav Lysne. Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis. In *Proceedings of the 11th ACM International Conference on Multimedia, ACM MM'03, Berkeley, California, USA*, pages 21–32, November 2003.

[34] Viktor S. Wold Eide, Frank Eliassen, and Olav Lysne. Supporting Distributed Processing of Time-based Media Streams. In Gordon Blair, Douglas Schmidt, and Zahir Tari, editors, *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA'01), Rome, Italy*, pages 281–288. IEEE Computer Society, September 2001.

[35] Viktor S. Wold Eide, Frank Eliassen, Olav Lysne, and Ole-Christoffer Granmo. Real-time Processing of Media Streams: A Case for Event-based Interaction. In Roland Wagner, editor, *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pages 555–562. IEEE Computer Society, July 2002.

[36] Viktor S. Wold Eide, Frank Eliassen, Olav Lysne, and Ole-Christoffer Granmo. Extending Content-based Publish/Subscribe Systems with Multicast Support. Technical Report 2003-03, Simula Research Laboratory, July 2003.

[37] Viktor S. Wold Eide, Frank Eliassen, and Jørgen Andreas Michaelsen. Exploiting Content-Based Networking for Video Streaming. In *Proceedings of the 12th ACM International Conference on Multimedia, Technical Demonstration, ACM MM'04, New York, New York, USA*, pages 164–165, October 2004.

[38] Viktor S. Wold Eide, Frank Eliassen, and Jørgen Andreas Michaelsen. Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming. In Surendar Chandra and Nalini Venkatasubramanian, editors, *Proceedings of the 12th Annual Multimedia Computing and Networking (MMCN '05), SPIE, San Jose, California, USA*, volume 5680, pages 155–166, January 2005.

[39] Viktor S. Wold Eide, Ole Christoffer Granmo, Frank Eliassen, and Jørgen Andreas Michaelsen. Real-time Video Content Analysis: QoS-Aware Application Composition and Parallel Processing. *Submitted to ACM Transactions on Multimedia Computing, Communications, and Applications, (TOMCCAP)*, April 2005.

[40] Ayman El-Sayed, Vincent Roca, and Laurent Mathy. A Survey of Proposals for an Alternative Group Communication. *IEEE Network*, 17(1):46–51, January/February 2003.

[41] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR)*, 35:114–131, June 2003.

[42] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking.*, 5(6):784–803, 1997.

[43] Aditya Ganjam and Hui Zhang. Internet Multicast Video Delivery. *Proceedings of the IEEE*, 93(1):159–170, January 2005.

[44] Paul Grace, Geoff Coulson, Gordon S. Blair, Laurent Mathy, Wai Kit Yeung, Wei Cai, David A. Duce, and Christopher S. Cooper. GRIDKIT: Pluggable Overlay Networks for Grid Computing. In Robert Meersman and Zahir Tari, editors, *CoopIS/DOA/ODBASE (2)*, volume 3291 of *Lecture Notes in Computer Science*, pages 1463–1481. Springer, 2004.

[45] Ole-Christoffer Granmo. *Toward Controlling Accuracy and Timeliness in Video Content Analysis*. Dr. Scient. thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, Norway, 2004.

[46] Mário Guimarães and Luís Rodrigues. A Genetic Algorithm for Multicast Mapping in Publish-Subscribe Systems. In *2nd IEEE International Symposium on Network Computing and Applications (NCA 2003), 16-18 April 2003, Cambridge, MA, USA*, pages 67–74. IEEE Computer Society, 2003.

[47] Intel Corporation. *Open Source Computer Vision Library. Reference Manual.*, 001 edition, December 2000. http://www.intel.com/research/.

[48] Hans-Arno Jacobsen, editor. *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, DEBS 2003, Sunday, June 8th, 2003, San Diego, California, USA (in conjunction with SIGMOD/PODS)*. ACM, 2003.

[49] Khronos Group. *An Overview of OpenML.*, 1_0 edition, April 2004. http://www.khronos.org/openml/.

[50] Charles Krasic. *A Framework for Quality-Adaptive Media Streaming: Encode Once — Stream Anywhere*. PhD thesis, OGI School of Science & Engineering at Oregon Health & Science University, February 2004.

[51] Jane W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[52] Jiangchuan Liu, Bo Li, and Ya-Qin Zhang. Adaptive Video Multicast over the Internet. *IEEE Multimedia*, 10(1):22–33, January/March 2003.

[53] Ketan Mayer-Patel and Lawrence A. Rowe. Exploiting temporal parallelism for software-only video effects processing. In Wolfgang Effelsberg and Brian C. Smith, editors, *Proceedings of the sixth ACM International Conference on Multimedia, September 13-16, 1998, Bristol, United Kingdom*, pages 161–169. ACM, 1998.

[54] Ketan Mayer-Patel and Lawrence A. Rowe. Exploiting spatial parallelism for software-only video effects processing. In Dilip D. Kandlur, Kevin Jeffay, and Timothy Roscoe, editors, *Multimedia Computing and Networking (MMCN'99), San Jose, California, USA*, volume 3654, pages 252–263. SPIE, 1999.

[55] Ketan Mayer-Patel and Lawrence A. Rowe. A multicast scheme for parallel software-only video effects processing. In John Buford, Scott Stevens, Dick Bulterman, Kevin Jeffay, and HongJiang Zhang, editors, *Proceedings of the seventh ACM International Conference on Multimedia, October 30 - November 05, 1999, Orlando, Florida, USA*, volume 1, pages 409–418. ACM, 1999.

[56] Steven McCanne, Martin Vetterli, and Van Jacobson. Low-Complexity Video Coding for Receiver-Driven Layered Multicast. *IEEE Journal of Selected Areas in Communications*, 15(6):983–1001, August 1997.

[57] Object Management Group. CORBA services, Event Service Specification, v1.1. *http://www.omg.org/*, 2001.

[58] Jens-Rainer Ohm. Advances in Scalable Video Coding. *Proceedings of the IEEE*, 93(1):42–56, January 2005.

[59] Wei-Tsang Ooi, Brian Smith, Sugata Mukhopadhyay, Haye Hsi Chan, Steve Weiss, and Matthew Chiua. The Dalí Multimedia Software Library. In Dilip D. Kandlur, Kevin Jeffay, and Timothy Roscoe, editors, *Multimedia Computing and Networking (MMCN'99), San Jose, California, USA*, volume 3654, pages 264–275. SPIE, 1999.

[60] Wei Tsang Ooi and Robbert van Renesse. Distributing media transformation over multiple media gateways. In Nicolas D. Georganas and Radu Popescu-Zeletin, editors, *Proceedings of the ninth ACM international conference on Multimedia, September 30 - October 05, Ottawa, Canada*, pages 159–168, 2001.

[61] Lukasz Opyrchal, Mark Astley, Joshua S. Auerbach, Guruduth Banavar, Robert E. Strom, and Daniel C. Sturman. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In Joseph S. Sventek and Geoff Coulson, editors, *Middleware*, volume 1795 of *Lecture Notes in Computer Science*, pages 185–207. Springer, 2000.

[62] Jörg Ott, Colin Perkins, and Dirk Kutscher. The message bus: A platform for component-based conferencing applications. In *CBG2000, the CSCW2000 workshop on Component-Based Groupware*, pages 35–42, December 2000.

[63] Burak Ozer and Wayne Wolf. Video Analysis for Smart Rooms. In *Internet Multimedia Networks and Management Systems, ITCOM, Denver Colorado USA*, volume 4519, pages 84–90. SPIE, July 2001.

[64] Peter R. Pietzuch and Jean Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In Jacobsen [48].

[65] Peter R. Pietzuch and Jean M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pages 611–618. IEEE Computer Society, July 2002.

[66] Tin Qian and Roy H. Campbell. Extending OMG Event Service for Integrating Distributed Multimedia Components. In Alvin P. Mullery, Michel Besson, Mário Campolargo, Roberta Gobbi, and Rick Reed, editors, *IS&N*, volume 1238 of *Lecture Notes in Computer Science*, pages 137–144. Springer, 1997.

[67] Hans Ole Rafaelsen and Frank Eliassen. Design and performance of a media gateway trader. In Robert Meersman and Zahir Tari, editors, *CoopIS/DOA/ODBASE*, volume 2519 of *Lecture Notes in Computer Science*, pages 675–692. Springer, 2002.

[68] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, pages 161–172, 2001.

[69] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-Level Multicast Using Content-Addressable Networks. In Crowcroft and Hofmann [26], pages 14–29.

[70] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Rachid Guerraoui, editor, *Middleware*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[71] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In Crowcroft and Hofmann [26], pages 30–43.

[72] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[73] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content Based Routing with Elvin4. In *Proceedings of AUUG2K, Canberra, Australia*, June 2000.

[74] Sergio D. Servetto, Rohit Puri, Jean-Paul Wagner, Pierre Scholtes, and Martin Vetterli. Video Multicast in (Large) Local Area Networks. In *Proceedings of IEEE INFOCOM, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 733–742, June 2002.

[75] Gary J. Sullivan and Thomas Wiegand. Video Compression – From Concepts to the H.264/AVC Standard. *Proceedings of the IEEE*, 93(1):18–31, January 2005.

[76] Sun Microsystems. *Java Media Framework, API Guide*, 2.0 edition, November 1999. http://java.sun.com/.

[77] Wim Taymans, Steve Baker, Andy Wingo, and Ronald S. Bultje. *GStreamer application development manual*, 0.8.8 edition, December 2004. http://gstreamer.freedesktop.org/.

[78] Wesley W. Terpstra, Stefan Behnel, Ludger Fiege, Andreas Zeidler, and Alejandro P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In Jacobsen [48].

[79] Chen-Khong Tham, Yuming Jiang, and Yung-Sze Gan. Layered Coding for a Scalable Video Delivery System. In *Packet Video 2003, Nantes, France*, April 2003.

[80] TIBCO Software. *TIBCO Rendezvous FAQ*, 2003. http://www.tibco.com/.

[81] Yao Wang, Zhu Liu, and Jin-Cheng Huang. Multimedia Content Analysis Using Both Audio and Visual Clues. *IEEE Signal Processing Magazine*, 17(6):12–36, November 2000.

[82] Yao Wang, Amy R. Reibman, and Shunan Lin. Multiple Description Coding for Video Delivery. *Proceedings of the IEEE*, 93(1):57–70, January 2005.

Part II

Research Papers

Paper I

Supporting Distributed Processing of Time-based Media Streams

Viktor S. Wold Eide, Frank Eliassen, and Olav Lysne

Published: In Proceedings of 3rd International Symposium on Distributed Objects and Applications (DOA'01), IEEE, pages 281-288, Rome, Italy, September 2001.

Evaluation: In total, 65 papers were submitted to DOA'01. Three program committee members, who are experts in one of the DOA'01 topics, reviewed each paper. As a result, 33 papers were accepted for publication.

Author Contribution: Eide was the driving force behind this article. The architecture was developed collectively in the project. Eide was the main investigator with respect to the media streaming, the filtering, and the feature extraction parts, as well as the issues related to interaction and synchronization. Eide also contributed to these issues with respect to the prototype.

Supporting Distributed Processing of Time-based Media Streams

Viktor S. Wold Eide, Frank Eliassen, and Olav Lysne
Department of Informatics, University of Oslo
Gaustadalleen 23, 0314 Oslo, Norway
{viktore, frank, olavly}@ifi.uio.no

Abstract

There are many challenges in devising solutions for on-line content processing of live networked multimedia sessions. These include content analysis under uncertainty (evidence of content are missed or hallucinated), the computational complexity of feature extraction and object recognition, and the massive amount of data to be analyzed under real-time requirements. In this paper we focus on middleware supporting on-line media content analysis. Our middleware supports processing, logically organized as a hierarchy of refined events extracted in real time from a set of potentially related time-based media streams. The processing can physically be distributed and redistributed during run time, as a set of interacting components, each performing some content analysis algorithm. The middleware is designed with reuse-ability, scalability, performance, resource management, and fault tolerance in mind by providing support for mechanisms such as, adaptation, reconfiguration, migration, and replication. The goal is to support applications in trading off the reliability and latency of the content analysis against the available computing resources.

1. Introduction

The technical ability to generate volumes of digital media data is becoming increasingly "main stream" in today's electronic world. 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. For example, an end-user may want to access live content in terms of high-level domain concepts under a variety of processing environments ranging from complex distributed systems to single laptops.

The focus of our research is the development of a framework for on-line (real-time) processing of networked multimedia sessions for the purpose of indexing and annotating the data being analyzed. An advantage of a carefully designed framework is that new sub-technologies of relevance for this kind of task can be plugged into the framework as they become available.

In this paper we focus on platform (or middleware) support for on-line media content analysis of networked media streams, executing media content queries as distributeable configurations of media content analysis algorithms. Most typically the processing can be logically organized as a refinement hierarchy of events where events higher in the hierarchy are aggregations of events at a lower level. An event corresponds to a particular kind of content detected in the media stream(s) (such as detecting movement or recognizing a particular shape or gesture in a video stream).

On-line analysis requires real-time processing of (live) networked sessions. Furthermore, automatic content-based multimedia indexing and retrieval is most normally based on a combination of feature extraction and object recognition. The reason for this is that it does not require manual entry of keyword descriptions. On the other hand the automatic indexing and annotation approach is computationally complex and only works satisfactory when limited to specific domains.

The above reasoning suggests that a distributed solution is required to cope with the computational complexity of feature extraction and object recognition, the massive amount of data to be analyzed in real time, and the scalability of the system with respect to the complexity of the session to be analyzed (e.g. the number of concurrent media streams) and the events to be detected. Additionally, a distributed solution may be more appropriate for problem domains having an inherent distributed nature, such as traffic surveillance where video cameras are distributed geographically.

Furthermore, to allow for real-time content-based access in a greater range of processing environments, the middleware should support processing to be continuously adaptable and scalable to the available processing resources. Such an approach allows the application to trade off between the reliability (probability of missed or hallucinated content detection) and latency of the content analysis (time required by the software to detect and report content). This

requires that a framework for on-line content analysis and access must be resource aware based on an open distributed resource model.

The contribution of the architectural framework represented by the middleware lies in the unique combination of media content analysis with QoS and resource awareness to handle real-time requirements.

This research is done in the context of the Distributed Media Journaling (DMJ) project[5] where the overall goal is to establish a framework for simplifying the task of building distributed applications for real-time content-based access to live media. The framework contains a set of general purpose media content analysis algorithms which may be selected for reuse by an application. The framework itself is extensible and also supports the addition of custom analysis algorithms. Applications are built by selection of some media processing algorithms which are then instantiated and deployed onto a set of different hosts.

The rest of the paper is organized as follows. In Section 2 we introduce and characterize our computational architecture for the purpose of deriving some requirements to middleware support. In Section 3 we discuss design issues of middleware supporting real-time content-based media access. Section 4 evaluates the adopted architectural principles with respect to the required properties of the middleware. In Section 5 we describe a first prototype of the DMJ framework and report on initial experiments. In Section 6 we discuss some related work, while we in Section 7 offer some conclusions and outlook to future work.

2. Architectural overview

A common way to build 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 Figure 1. At the lowest level of the hierarchy media streams are filtered and transformed, such as transforming a video stream from color to black and white only, reducing spatial or temporal resolution, etc. The transformed media streams are then fed to feature extraction algorithms. Feature extraction algorithms operate on samples or segments (a window of samples) from the transformed media streams and calculate features such as texture coarseness, center of gravity, color histogram, etc. Results from feature extraction algorithms are generally reported to classification algorithms higher up in the hierarchy that are responsible for detecting higher level domain concepts such as a “person” occurring in a media stream.

In our architecture, the media processing hierarchy is similar, but the different algorithms are now encapsulated in components - F (Filter), E (feature Extraction), and C (Classification) components, as illustrated in Figure 1. The

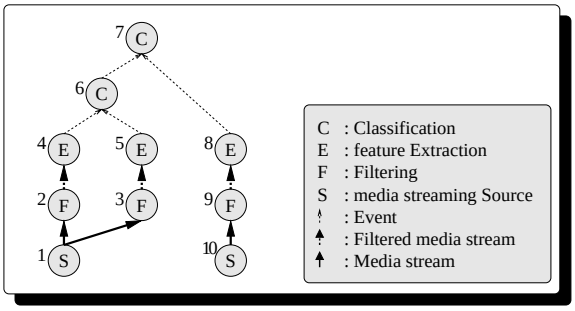

Figure 1. A content analysis hierarchy example.

content analysis task is realized as a collection of (potentially distributed) interacting components, where components monitor other components and react to particular changes in their state.

In the DMJ computational model, F, E, and C components have multiple interfaces. Event based interfaces consume or generate events of specified types. Stream interfaces consume media streams of specified formats. C components have event based interfaces only, E components have both event based and stream based interfaces while F components have stream based interfaces (the event based interfaces of F components, illustrated in Figure 2, is for management purposes). When configuring a set of F, E, and C components such as in Figure 1, bindings (or communication associations) are established between sets of interfaces of the associated components. Bindings can be one-one such as the binding between components 8 and 9 in Figure 1, one-many such as the binding between interfaces of components 1, 2, and 3, many-one such as the binding between interfaces of components 4, 5, and 6, and many-many (not illustrated in Figure 1).

A C component consumes events produced by other E and/or C components and generates events that again may be consumed by other C components, reported directly to a user (e.g. as alarms), or stored in a database as indexing information (meta-data). The DMJ middleware does not make any assumptions as to what method is applied in C components for detection of higher level domain concepts. However, in Section 5 we report on experiences with a prototype application in which the construction of C components is based on dynamic object-oriented bayesian networks and particle filters[7].

The DMJ middleware provides generic services maintaining an extensible repository of F, E, and C components and for selecting registered components based on the required media processing function or higher level domain concept, the cost of the algorithm and its reliability characteristics.

3. Design

In this section we will look at design issues, focusing on time and synchronization, the interaction model and configuration.

3.1. Time and synchronization

The task of a C component is basically to determine whether a higher level event has taken place, based on input from multiple E and/or C components. This raises questions related to the perception of time in the different components, and the definition of duration of events.

The communication between the E and C components is designed as exchange of *Event Descriptors*. Each Event descriptor can be viewed as tuples on the form `<eStart, eEnd, eType, eInstance>`. The two first elements indicate that we allow events to have a duration in time, thus we support a wide class of sustained actions to be interpreted as events. Both the event type and the event instance are made explicit, to allow filtering of interesting events based on both type and value. Although events relating to multiple time-intervals are imaginable, we will not consider such events in this context. The aggregation of simple sustained events into composite sustained events is performed by C components, according to a temporal specification. Different methods include *interval*, *axes*, *control flow*, *event*, and *script* based synchronization specifications[19]. The information provided in the Event Descriptors is designed to support such specifications. Interval-based methods [8, 1, 22] allows specifications such as *e1 before e2* and *e1 while e2* for events *e1* and *e2*.

We have based our design upon a common knowledge of global time in all components. It is well known that in a distributed system, global time can only be available down to certain levels of accuracy. The most widely used global time service in the Internet is the Network Time Protocol (NTP), as specified in RFC 1305[11]. Synchronization accuracies between different hosts is reported to be in the order of tens of milliseconds on the Internet in general and one millisecond over LANs[12]. The required level of accuracy in our setting is application dependent, but if we consider video streams with 25 frames per second, NTP may provide frame level accuracy even over wide area networks.

The time stamping of each media stream is done as close to the source of the stream as possible. In most cases this coincides with the first intelligent, NTP aware, node that the stream passes through (e.g. the computer to which a video camera is connected). Hence the worst case skew in the perception of time in each media stream will be the sum of the worst case skew from NTP, and the worst case delay skew in time stamping the media stream. The former component is well known through published measurements, and the latter can for most relevant applications be measured in advance. The remaining skew must be handled by the C components by introducing intervals of uncertainties in the definition of interval-based operations such as *before*, *after*, and *while*. This will translate the time skew of the system into a minor increase in either lost or hallucinated events, according to an overall system policy.

3.2. Interaction model

The analysis task is realized as a collection of interacting components where some components monitor other components and react to particular changes in their state.

An interaction model that enables multiple components to share results generated by other components is desirable from a resource consumption viewpoint, for scalability reasons. The interaction model should allow *adaptation*, the updating of some state in a component to better fit the current environment and stage of computation, such as decrementing the value of a variable to reduce CPU usage in an overload situation. Runtime removal, addition or substitution of components, *reconfiguration*, should also be allowed, substituting a less reliable E component by a more reliable one, trading improved event detection reliability for CPU cycles. Being able to move a running component from one host to another, *migration*, is also a useful mechanism, moving components away from loaded hosts (load-balancing) or hosts being taken down for maintenance purposes. To achieve such functionality, it would be advantageous that the different components do not need to be aware of each other, but communicate indirectly.

The outlined requirements and communication patterns fit very well with the publish/subscribe interaction paradigm, leading to an event based model. Event based systems rely on some kind of event notification service, such as an event (message) broker. The responsibility of the event broker is to propagate events (messages) from the event producer to event consumers residing in different computers, generally in a many-many manner. Figure 2 illustrates the logical coupling of components, where the event broker acts as a level of indirection. In addition to F, E, and C components an *End Consumer* component has been drawn in the figure. This component represents the final destination for the meta-data extracted from the media streams, such as a database or a user interface component.

Another important aspect regarding both the interaction model and resource consumption is *push* versus *pull* style communication. Events produced by components at the bottom of the hierarchy are likely to be interesting to a large number of other components. Push style interaction fits one-many communication well, and the need for an explicit request message, introducing delay, is eliminated. From

a resource consumption viewpoint, an approach where all components perform calculations and push these results continuously, is not very attractive. In general, a component is only interested in results from another component in certain cases. As an example, consider a C component, *C3*, producing events, *e3*, according to a specification such as *e1 before e2*, where *e1* is generated by component *C1* and *e2* is generated by component *C2*. In this case *C3* is not interested in events from component *C2* before an event *e1* from *C1* has been received. In this case component *C3* may reduce overall resource consumption by explicitly requesting, pulling, results from component *C2*. Both push and pull style interaction have their advantages, but in different situations. The suitability of pull versus push style interaction may also change during time. As an example, a component working in pull mode may enter push mode for some time.

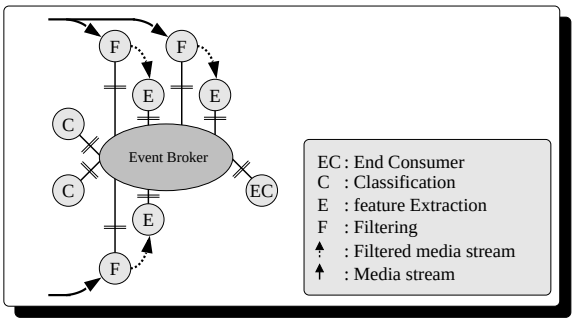

Figure 2. The event broker introduces a level of indirection between components.

Some algorithms are very complex and time consuming and therefore not able to process each media sample in real-time. From a resource consumption viewpoint it is important to execute such algorithms only when absolutely necessary and from a reliability viewpoint it is important to activate such algorithms only for particularly interesting samples. This reasoning suggests that such algorithms should be demand driven, pulled by other components. In [7] a method for constructing C components is described which allows a C component to pull E components in a "hypothesis" driven fashion - when the information gain is expected to be significant compared to the computational cost. This method is particularly interesting for these very complex and time consuming algorithms.

A further refinement of pull mode interaction is the distinction between *eager* and *lazy* components. A lazy component may save host computer resources by not doing any calculations until explicitly requested. An eager pull mode component on the other hand, is able to deliver a result immediately upon request, but at the expense of increased resource consumption.

In general, the choice between push/pull and lazy/eager is an optimization problem which depends on different characteristics, such as the computational environment (both static and dynamic), the components and how they are distributed, the media content, the acceptable latency, and the desired level of resource utilization.

3.3. Configuration

The configuration phase of an application for distributed processing of media streams is handled in several stages. A set of functions for fulfilling the media processing task is selected and for each function an appropriate implementation is chosen. The details of this process are not discussed any further in this paper and we assume that the set of F, E, and C components for a specific analysis task has been selected.

The lowest level of configuration must deal with binding of producer and consumer interfaces of F, E, and C components, as well as connecting media streams to F and E components. The binding of a media stream to a F or E component is straightforward, as each F and E component is supposed to be able to handle one medium (e.g. video) in one or several formats. The details of this binding might include other components responsible for sending/receiving the stream to/from the network and stream encoding/decoding.

The binding of event consumer and event producer interfaces is supported by an Event-Type Repository. An event interface is specified as the event type it produces or consumes. The event type is interpreted as a reference into the Event-Type repository, indicating the type of events generated by the producer as expected by the consumer. Compatibility check is done at binding time simply by checking that the producer and consumer interfaces refer to the same Event-Type.

4. Evaluation

Desired framework characteristics, such as reuse-ability, scalability, performance, resource utilization, and fault tolerance, are influenced by the degree by which the design of the framework supports the following underlying mechanisms:

- *Modularization:* At one extreme, an analysis task could be implemented and linked into one binary, executing as a single process. Such an approach is not modular and would make both implementation and reuse difficult. Our approach is component based, allowing new components to be plugged in whenever available.

- *Distribution:* Although a single computer may host all components of a media processing task, the computational complexity and the massive amounts of data associated with real-time processing of time-based media streams require a distributed solution. Figure 3 is a further refinement of Figure 1 and illustrates the deployment of components onto different computers. The design allows the incremental addition of new processing nodes and network technology for improved scalability, giving a certain degree of scalability transparency. NTP is a solution to the clock synchronization problem at the same time as being a worldwide distributed and scalable service. Hosts synchronization and component interaction are both illustrated logically in the figure.

- *Adaptation:* The complexity of the logic embedded inside a component increases with the components adaptability. Components may themselves monitor and adapt to environmental changes, but information may not be locally available or the decision is best taken by other components. In the last case, an interface for receiving adaptation requests from other components is necessary. The event broker can be used to deliver such adaptation requests.

- *Reconfiguration:* Adaptation might be sufficient for handling small changes, while reconfiguration, having more overhead, has the potential of handling larger changes. Adaptation is therefore used on a smaller time-scale than reconfiguration. Support for reconfiguration is partly handled in our design, by the event broker which makes communication between components both access and location transparent.

- *Migration:* The level of indirection between components simplifies migration, but communication is not the only aspect relevant in this context. From a migration viewpoint, stateless components are preferable because otherwise state has to be maintained during migration. In general this might require special code in the component itself. The design of the framework in this respect support migration transparency to a certain extent.

- *Replication:* The design also allows components to be replicated at a number of hosts for improved performance, increased availability and fault tolerance. In a simple approach, a certain degree of failure transparency can be obtained by filtering identical events, received from different replicas.

We believe that the design of the framework is flexible and distribution transparent to a certain extent, allowing new sub-technologies to be plugged into the framework as they become available.

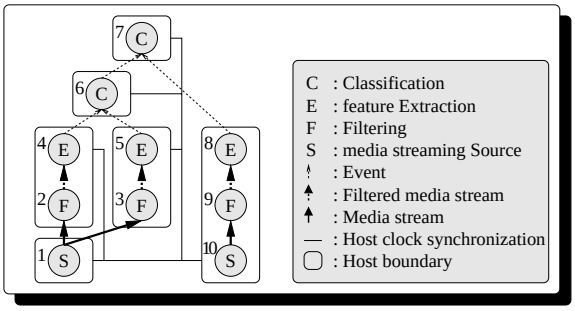

Figure 3. An example of distributed processing of the configuration in Figure 1.

5. Prototype

The current implementation of a prototype consists of the event broker, a component repository and a test application, each described in this section. This prototype operates on a "best effort" basis.

5.1. Event broker

Important requirements for an event notification service supporting distributed processing of time-based media streams are:

- The event notification service must allow bindings of different causalities, support both push and pull style interaction and provide a level of indirection to simplify adaptation, reconfiguration, migration and replication. Additionally, the service must be realized as a distributed and scalable service, balancing requirements for real-time communication and low event propagation delay against ordering and reliability guarantees associated with event delivery.

Different event notification service technologies, such as CORBA Event Service[14], CORBA Notification Service[13], SIENA[3] (Scalable Internet Event Notification Architecture), and somewhat related, shared spaces approaches such as JavaSpaces[21] are available, but the event broker in this prototype is based on Mbus[16].

Mbus is a standard[17] currently being worked out by the IETF. Mbus is designed to support *coordination and control* between different application entities, corresponding roughly to components in our terminology. The standard defines message addressing, transport, security issues and message syntax for a lightweight message oriented infrastructure for ad-hoc composition of heterogeneous components. The authors of Mbus state that Mbus is not intended for use as a wide area conference control protocol, for security (conforming Mbus implementations must support both

authentication and encryption[17]), scalability, message reliability and delay reasons[16]. In the following we evaluate the suitability of Mbus as an event broker, with respect to the requirements given above.

Mbus supports binding of different causalities by using a "broadcasting" and filtering technique. All components participating in a specific Mbus session subscribe to an IP multicast address and in effect Mbus messages are "broadcasted" to the set of computers hosting components participating in this Mbus session. The Mbus layer in each component sees all Mbus traffic and must filter messages. Filtering as close to the source as possible is beneficial from a resource consumption viewpoint and Mbus is rather suboptimal in this respect. As an optimization, several Mbus sessions may be started, using a set of IP multicast addresses. Each component participate in a subset of these Mbus sessions and messages are sent to different sessions based upon some predefined scheme. Important for filtering is the addressing used in Mbus. The address of a component is specified when initializing the Mbus layer. The Mbus header includes source and destination addresses, each a sequence of attribute-value pairs, of which exactly one pair is guaranteed to be unique (combination of process identifier, process demultiplexer and IP address). Each Mbus component receives messages addressed to any subset of its own address. A Mbus component is able to address a single, *"(id:7-1@129.240.64.28)"*, a subset, *"(module:pd media:video)"*, or all, *"()"*, Mbus components by specifying an appropriate sequence of attribute-value pairs. As a result, bindings between components are implicit.

It should by now be evident that Mbus supports push based interaction. Some higher level Mbus services are described in [9], such as abstractions for remote procedure call. Pull style interaction is achieved by either sending requests as event notifications or by using the remote procedure call service.

An Mbus based event broker acts as a layer of indirection between components giving both access and location transparency simplifying reconfiguration and migration. Component awareness is supported by a soft state approach, where the Mbus layer listens and periodically sends self-announcements messages on behalf of its component. When migrating a component to another host, its Mbus address remains the same (except for the value of the *id* attribute, reported in succeeding self-announcements).

Regarding scalability, message propagation delay and reliability of event delivery, an Mbus based event broker inherits many of its characteristics from IP multicast[2], which is realized as a distributed and scalable service. The state necessary for forwarding IP multicast packets is calculated and stored in both routers and in hosts acting on behalf of multicast receivers in a distributed fashion. The Mbus component awareness functionality limits scalability, but the rate of self-announcements is adapted to the number of entities participating in a session.

IP multicast also decreases latency which is very important for the domain initially targeted by IP multicast, real-time, high bandwidth multi-user applications, such as video and audio conferences.

At the transport level, Mbus messages are encapsulated in UDP packets, transported unreliably by IP multicast. In the special case where the message is targeted at exactly one receiver, reliable unicast delivery is supported by the Mbus layer, using acknowledgement, timeout and retransmissions mechanisms. In general reliable delivery does not fit very well with applications having real-time properties and even worse when considering multicast delivery and ordering requirements. Our approach is to handle the unreliability of event delivery in the same way as we handle the unreliability of analysis algorithms, which may fail to detect and report an event or report false positives. The Mbus/UDP/IP multicast protocol stack does not give any ordering guarantees, but assuming global time and associating a time interval to each event (see section 3.1) handles this ordering problem, except for very time-sensitive applications.

From the discussion above, we believe that Mbus is a good alternative as an event broker. From a prototyping viewpoint it is easy to integrate, requiring few lines of code, and the text based message format simplifies message snooping.

5.2. Component repository

In the current prototype of the framework, ways to implement F, E, and C components have been identified. A component repository, where components are selected manually, has been realized. Handling the diversity and complexity of different media formats, such as audio and video, is beyond the scope of our research. The F and E components are realized using the Java Media Framework[20]. JMF performs low level media tasks, such as capture, transport, streaming, (de)multiplexing, (de)coding, and rendering. JMF also provides a pluggable architecture for integrating custom media processing algorithms. The F and E components developed for the prototype are implemented as classes in the Java programming language and pluggable into the JMF framework. F and E components implement a method (e.g. iterating through all pixels of a video frame performing some calculations) which is invoked when a new media sample is available. The C component implemented for this prototype is based on dynamic object-oriented Bayesian networks (a generalization of the hidden Markov model) and particle filters[7]. In the following section we will describe a few of the implemented components.

5.3. Test application

The purpose of the test application is to gain experience, serve as a proof of concept, and to verify the flexibility of Mbus with respect to bindings of different causalities as well as the requirements listed in section 5.1.

The chosen test case is an implementation of a video segmentation application, where the goal is to detect and associate meta-data information, such as *motion* and *cut* (scene change), with a streamed real-time video. The application is composed of a media streaming source, two E components and a C component:

- Media streaming source - streams video from a file or from a capture card connected to a video camera. We used applications such as vic[10] and JMStudio (bundled with the JMF) for this purpose. The protocol stack was MJPEG/RTP[18]/UDP/IP multicast.
- E component *PixelValueDiff* - adds together the pixel value difference between the previous and the current frame. Object or camera movement between two consecutive frames causes a larger difference value being reported.
- E component *ColorHistogramDiff* - calculates the difference between a color histogram for the previous frame and a color histogram for the current frame. Moving objects or camera motion should not affect the difference being reported, but cut (scene changes) most likely will.
- C component *SceneChangeClassifier* - receives results from the E components and correlates the results from each E component and reports *motion* and *cut*.

In this test application we do not use any custom F components. However the JMF runtime system configures a flowgraph including components for receiving the video stream from the network, decoding, and transforming the video data into a format supported by our E components.

The bindings between the different components are similar to the bindings between component 1, 2, 3, 4, 5, and 6 in Figure 1 (where component 2 and 3 represents the filtering performed by the JMF runtime system).

Component interaction uses push style communication. The video source application pushes (multicasts) the video stream over the network. The stream is received by two different processes, each executing the JMF runtime system and hosting one of the E components. In each process, the JMF runtime system invokes the media processing method of the E component whenever a new frame arrives. Each E component pushes the calculated results encapsulated in messages over the event broker to the C component which is hosted by a separate process.

The deployment of components onto hosts is performed manually in this prototype. The flexibility of the Mbus based event broker was confirmed by experiments - configuration and compilation was unnecessary before starting components on different hosts or when reconfiguring the media processing task. Mbus performs as expected, both as an intra-host event broker, but also when components are hosted by different computers interconnected by a local area network. Some successful migration experiments have also been conducted[15], using Voyager[6] for moving running components between hosts.

We have performed some preliminary experiments, testing different distribution strategies. On one extreme, all components executed on a single machine while on the other extreme each component was hosted by a different computer. The results show that distributing components to different computers reduces the average load on each host. This allows the development of more resource demanding, but also more reliable components, improving scalability.

6. Related work

The semantic multicast service, described in [4], is an approach for dynamic information sharing, designed to support effective dissemination of the information produced in collaborative sessions, such as a video conference. The design describes a network of proxy servers which gather, annotate, filter, archive and analyze (both on-line and off-line) media streams to better suit the needs of different users at the right amount of detail. Users specify their needs in a profile, used during subscription. Semantic multicast adds an additional level of indirection to IP multicast, used to transport information and control messages. The paper assumes that the meta-data for the streams (and profiles) have standardized formats and have been generated, either manually or automatically. Our framework could be used for generating such meta-data by performing real-time processing of time-based media streams transported by IP multicast.

Section 5.1 listed some event notification service technologies, among others the SIENA[3] (Scalable Internet Event Notification Architecture). The goal of the SIENA research project is to design and build an Internet scale event notification middleware service. SIENA is implemented as a distributed network of servers, some providing an access point for clients. Access points receive subscriptions from consumers expressing their interest in events, and advertisements from generators about potentially notifications. SIENA is responsible for selecting and delivering events matching such subscriptions. The main challenge faced is to balance expressiveness in the selection mechanism against the scalability of event delivery. SIENA provides two mechanisms for selection of notifications, *filters* and *patterns*. A filter is a specification of attributes and constraints on their

values, while a pattern is syntactically a sequence of filters. Selections not expressible in the SIENA model, must be handled at the application level. SIENA is designed as a "best-effort" service, but assumes the existence of a global clock, timestamping events to detect and account for latency effects. This is similar to our approach, described in section 3.1. We think that SIENA is an interesting event broker candidate in a future prototype.

7. Conclusions and future work

In this paper we have presented an approach for developing middleware supporting distributed processing of time-based media streams. We describe a logical processing architecture as a hierarchy of aggregated events. We argue that a distributed solution is required to cope with the computational complexity and the massive amounts of data to be handled under real-time requirements. Additionally, a distributed solution is often more appropriate for problem domains having an inherent distributed nature. The framework is designed with distributed processing and resource management in mind and much attention has been paid to required underlying mechanisms such as adaptation, reconfiguration, migration, and replication. A prototype of the framework as well as a test application have been implemented, serving as a proof concept and for evaluation purposes. We find the results promising.

In our future work we will conduct further experiments to identify areas with potential for improvement. We will target resource management and QoS issues specificly. Important from a resource management viewpoint is the use of pull style interaction, where host resources are not consumed when not absolutely necessary.

8. Acknowledgments

We would like to thank all persons involved in the Distributed Media Journaling project for contributing to ideas presented in this paper. We also would like to thank the reviewers for valuable comments.

The DMJ project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431. In addition, some of the technical equipment has been sponsored by the Department of Informatics, at the University of Oslo, Norway.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] K. C. Almeroth. The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment. *IEEE Network*, 2000.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, Portland OR, USA, July 2000.

[4] S. K. Dao, E. C. Shek, A. Vellaikal, R. R. Muntz, L. Zhang, M. Potkonjak, and O. Wolfson. Semantic Multicast: Intelligently Sharing Collaborative Sessions. *ACM Computing Surveys*, June 1999.

[5] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Distributed Journaling of Distributed Media. In *Norsk Informatikkonferanse*, pages 31–42, available from http://www.ifi.uio.no/~dmj/, 2000.

[6] G. Glass. Voyager - the universal orb. Technical report, Objectspace, January 1999.

[7] O.-C. Granmo, F. Eliassen, and O. Lysne. Dynamic Object-oriented Bayesian Networks for Flexible Resource-aware Content-based Indexing of Media Streams. *Scandinavian Conference on Image Analysis, SCIA'2001*, 2001.

[8] C. Hamblin. Instants and intervals. In *Proceedings of 1st Conference on the International Society for the Study of Time*, pages 324–331, 1972.

[9] D. Kutscher. The Message Bus: Guidelines for Application Profile Writers. *Internet Draft , draft-ietf-mmusic-mbus-guidelines-00.txt*, 2001.

[10] S. McCanne and V. Jacobsen. Vic: A flexible Framework for Packet Video. *In ACM Multimedia'95, pp. 511-522*, 1995.

[11] D. L. Mills. Network Time Protocol (version 3). Specification, Implementation and Analysis. *RFC 1305*, 1992.

[12] D. L. Mills. Improved Algorithms for Synchronizing Computer Network Clocks. *IEEE Transactions Networks*, pages 245–254, 1995.

[13] Object Management Group Inc. CORBA services, Notification Service Specification, v1.0. *http://www.omg.org/*, 2000.

[14] Object Management Group Inc. CORBA services, Event Service Specification, v1.1. *http://www.omg.org/*, 2001.

[15] R. W. Olsen. Component Framework for Distributed Media Journaling. Master's thesis, (in Norwegian), Department of Informatics, University of Oslo, May 2001.

[16] J. Ott, D. Kutscher, and C. Perkins. The Message Bus: A Platform for Component-based Conferencing Applications. *CSCW2000, workshop on Component-Based Groupware*, 2000.

[17] J. Ott, C. Perkins, and D. Kutscher. A message bus for local coordination. *Internet Draft , draft-ietf-mmusic-mbus-04.txt*, 2001.

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobsen. RTP: A Transport Protocol for Real-Time Applications. *RFC 1889*, 1996.

[19] R. Steinmetz and K. Nahrstedt. *Multimedia: Computing, Communications & Applications*. Prentice Hall, 1995.

[20] Sun Microsystems Inc. Java Media Framework, API Guide, v2.0. *http://java.sun.com/*, 1999.

[21] Sun Microsystems Inc. JavaSpaces Service Specification, v1.1. *http://www.java.sun.com/*, 2000.

[22] T. Wahl and K.Rothermel. Representing time in multimedia systems. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 538–543. IEEE Computer Society Press, 1994.

Paper II

Real-time Processing of Media Streams: A Case for Event-based Interaction

Viktor S. Wold Eide, Frank Eliassen,
Olav Lysne, and Ole-Christoffer Granmo

Published: In Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), IEEE, pages 555-562, Vienna, Austria, July 2002.

Evaluation: In total, 43 papers were submitted to DEBS'02. The review process was double blind. The paper was reviewed by three persons. As a result, 18 full papers and 6 short papers were accepted for publication.

Author Contribution: Eide was the driving force behind this article and the principal researcher. The coauthors contributed to the presented ideas through discussions and by commenting on different draft versions of the paper. With respect to the described prototype, Granmo implemented the C-components, while Eide did the rest of the implementation.

Real-time Processing of Media Streams: A Case for Event-based Interaction

Viktor S. Wold Eide[1,2], Frank Eliassen[1,2], Olav Lysne[1,2], and Ole-Christoffer Granmo[1,2]
{viktore, olegr}@ifi.uio.no, {frank, olavly}@simula.no

[1]Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0314 Oslo, Norway

[2]Simula Research Laboratory
P.O. Box 134, N-1325 Lysaker, Norway

Abstract

There are many challenges in devising solutions for on-line content processing of live networked multimedia sessions. These include the computational complexity of feature extraction and high-level concept recognition, the massive amount of data to be analyzed under real-time requirements and the intricate correspondence between low-level features and high-level concepts. Our approach to these challenges is a distributed architecture consisting of interacting components encapsulating feature extraction and concept classifier algorithms. The purpose of the framework is to simplify the development of applications for the domain of on-line multimedia content processing.

In this paper we focus on the architecture of the framework and argue that it fits well to the publish / subscribe interaction paradigm, leading to an event-based interaction model. Furthermore, we analyze different aspects of the application domain in more depth, such as requirements for scalability, reconfiguration, migration, event notification selection, filtering, and ordering. The main contribution of this paper is, that we for each aspect show how a suitable event notification service may satisfy the corresponding requirements. We also describe parts of a framework prototype. In particular we report on how the event notification service used satisfies the identified requirements.

1. Introduction

The technical ability to generate volumes of digital media data is becoming increasingly "main stream" in today's electronic world. On the other hand, technology for automatic indexing (associating meta-data to) such media data is immature.

The main challenges that must be addressed include the computational complexity of feature extraction and high-level concept recognition, and the massive amount of data to be analyzed under real-time constraints. By taking advantage of a parallel processing architecture, features can be extracted in parallel. A multi agent based system for course-grained distribution of feature extraction is presented in [12]. Another challenge faced by real-world applications is the noise introduced into the extracted features (e.g. shadows in a video). In [8], a generic automatic video surveillance system is described, for recognizing various kinds of human activities. A statistical approach (Bayesian network) is used for noise suppression.

In our research we are developing a component based framework with the goal of simplifying the development of distributed scalable applications for on-line media content analysis. The framework is a generic modular application which is instantiated during the development of specific content analysis applications. An advantage of a carefully designed framework is that new sub-technologies can be plugged into the framework as they become available. As an example, third parties may extend the framework by providing new and/or better feature extraction algorithms.

The focus of this paper is on the requirements for communication and distribution of the content analysis framework. Based on our analysis of different aspects of the application domain, such as requirements for scalability, reconfiguration, and migration, we show that requirements for communication and distribution is better supported by a suitable distributed event notification service rather than by a synchronous communication service such as RMI. Furthermore, our experiments with a framework prototype approve this conclusion from the analysis.

The rest of the paper is organized as follows. In Section 2 we describe the architecture of typical content analysis applications. In Section 3 we introduce our computational architecture and claim that it fits well to the publish / subscribe interaction paradigm. The rest of this section is devoted to various aspects of our application domain and the requirements imposed upon the event notification service. In Section 4 we describe those parts of a prototype of the framework which are relevant from an event notification service point of view. In Section 5 we offer some conclusions and outlook to future work.

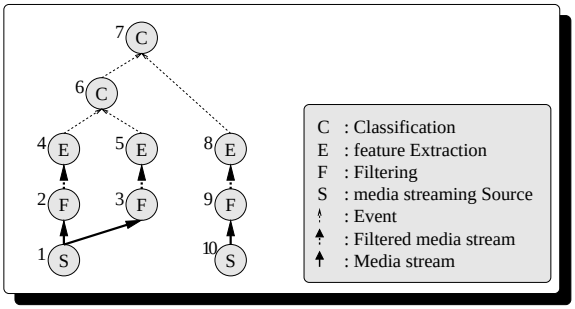

Figure 1. An example of a content analysis hierarchy.

2. Content Analysis

A common way to build 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 Figure 1. At the lowest level of the hierarchy there are media streaming sources. At the level above, media streams are filtered and transformed. The transformed media streams are then fed to feature extraction algorithms. Feature extraction algorithms operate on samples/segments from the transformed media streams and, in case of video, calculate features such as texture coarseness, center of gravity, color histograms, and motion vectors. Results from feature extraction algorithms are generally reported to classification algorithms higher up in the hierarchy that are responsible for detecting higher level domain concepts such as a "person" occuring in a media stream. In other words, classification is interpretation of extracted features in some application specific context.

Typically, 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. In the following section we present a framework developed in the DMJ project[4], addressing these weaknesses.

3. Application domain and Event Notification Service Requirements

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 are now encapsulated in components - F (Filter), E (feature Extraction), and C (Classification) components. The content analysis task is realized as a collection of interacting components, where components indirectly monitor other components and react to particular changes in their state.

As we will see in the following, the requirements for our application domain and the communication patterns 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. The responsibility of the event notification service is to propagate events from the event producers to event consumers, generally in a many-many manner.

We will now characterize and describe application requirements and see how each of these translates into requirements for the event notification service.

3.1. Event-based component interaction

From the example in Figure 1 it should be clear that components interact in different ways, having different causalities such as - one-one, one-many, many-one, and many-many (not illustrated). In this paper we mainly focus on the interaction between E and C components, which is designed as exchange of event notifications.

One-many communication, often used for sharing results between a number of components, is important for resource consumption and hence scalability reasons. If a component has high data rate input, spends a lot of time processing, has relatively low data rate output, and the output is of interest to a number of other components, then it is a good candidate for sharing.

The causality of the interaction as well as the loose coupling between components are some of the arguments for en event-based interaction model, illustrated in Figure 2. The "End Consumer" component in the figure represents the final destination for the meta-data extracted from the media streams, such as a database or a user interface component. In our framework, components are the unit of distribution.

3.2. Distribution

The framework must support distributed processing to cope with the massive amount of data to be analyzed in real-time and the computational complexity of feature extraction and concept recognition. Scalability is important along several axes, including the complexity of the content to be recognized and the number of media streams concurrently analyzed. Additionally, a distributed solution may be more appropriate for problem domains having an inherent distributed nature. As an example, in a traffic surveillance application video cameras are distributed geographically. Processing of video data in a host located close to a camera reduces network bandwidth consumption. As a

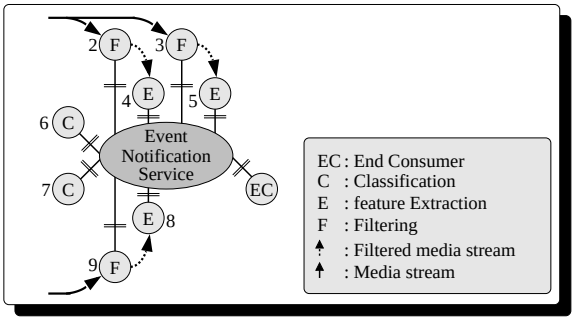

Figure 2. The event notification service introduces a level of indirection between components.

consequence, the event notification service should be able to operate across wide area networks.

On the other hand, parts of an application may consist of a number of tightly coupled components, configured to execute inside a single host. Even more tightly coupled components may execute inside a single process, in order to exploit data locality even further and avoid copying large amounts of data between different address spaces. Other cases requiring intra host and/or intra process component interaction include situations where the runtime environments available for the media processing task is limited to a single computer. From our framework perspective, the event notification service should handle such cases too, in order to simplify application development and offer consistency with respect to both programming and execution.

A distributed event notification service is also required for scalibility and in order to avoid a single point of failure.

Figure 3 illustrates how the set of components may be deployed onto a set of hosts. As can be seen, some of the computers host several components. Neither the distribution of the event notification service itself nor the process boundaries are illustrated in the figure, but as described, a number of components may share a single address space.

3.3. Resource management

Common processing environments do not support resource reservation of CPU, memory, network bandwidth, etc. and can not give any guarantees beyond "best effort". As a consequence, the available resources change dynamically over time. A component executing in this kind of environment may experience overflow situations when it is not able to perform processing within the limited time frame, determined by the real time properties. Similarly, underflow situations may occur if the network is overloaded, causing starvation at components. The infrastructure is also dynamic, although on a different time scale. It undergoes evolution where computers, cameras, sensors, etc. are added and removed. Handling such changes gracefully is important, especially for large scale, continuously running applications. Adaptation might be sufficient for handling small changes on a relatively short time scale, while reconfiguration, having more overhead, has the potential of handling larger changes. The level of indirection between components introduced by an event-based interaction model simplifies both reconfiguration and migration, also described in [5]. However, a resource management part of the framework must know the hosts and which components are currently executing at each, in order for reconfiguration and migration to be meaningful.

An approach where *all* components process and push results continuously is not always suitable, although it fits periodic processing of time-based media and one-many communication well. Some components are interested in results from other components only in certain cases and a demand driven event distribution is more appropriate from a resource consumption viewpont. As an example, consider a C component, *C3*, producing events, *e3*, according to a specification such as *e1 before e2*, where *e1* is generated by component *C1* and *e2* is generated by component *C2*. In this case *C3* is not interested in events from component *C2* before an event *e1* from *C1* has been received. In this case component *C3* may reduce overall resource consumption by explicitly requesting, pulling, results from component *C2*. A hybrid between push and pull is also possible, such as *push for n seconds*. The most suitable style may change during runtime. The event notification service should allow components to dynamically select the appropriate operation point from the push/pull spectrum. In [7], a method for constructing C components is described which allows a C component to pull E components in a "hypothesis driven" fashion - when the information gain is expected to be significant compared to the computational cost.

3.4. Event notifications

Our approach for specifying event notification types is to use a sequence of type, name (and value) tuples. As an example, an E component may generate event notifications containing the following types and names:

```
string  media_type
string  media_source
string  function
string  component_type
float   motion
time    start
time    stop
```

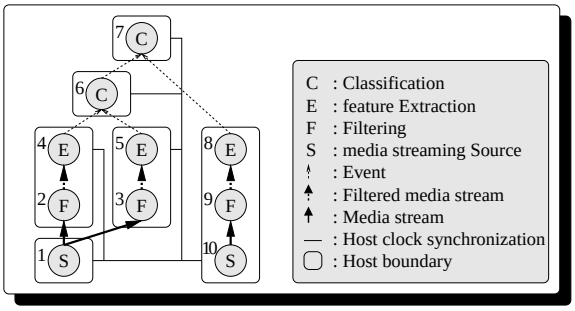

Figure 3. An example of distributed processing of the configuration in Figure 1.

The purpose of the *motion* variable is to represent a quantization of the level of motion, calculated by an E component on the basis of two consecutive video frames. The variables *start* and *stop* represent the interval in which the event, reported in an event notification, took place. An instance of such an event notification may look something like:

```
string  media_type      video
string  media_source    rtp://224.0.7.1/1111
string  function        motion_estimation
string  component_type  E
float   motion          0.32
time    start           3215927128.020
time    stop            3215927128.060
```

3.5. Event selection and filtering

Following the example above, a C component may subscribe to event notifications generated by E components processing video from a specific source, performing motion estimation, but limited to event notifications where the motion is above some threshold, by supplying the following filter:

```
string  media_type      video
string  media_source    rtp://224.0.7.1/1111
string  function        motion_estimation
string  component_type  E
float   motion          > 0.5
```

The effect of filtering depends on where the filtering takes place. At one extreme, the code responsible for event notification filtering is supplied as a library and linked into the component code. In this case the event notification service "broadcasts" all event notifications to all components which perform filtering by executing this library code. The scalability of this approach is very limited, since all network components (routers, switches), all hosts (their network interface cards and operating systems) and all processes (the library code) see all events. At the other extreme, the event notification service is implemented as a distributed service, either provided as a native and ubiquitous service of the network itself or alternatively added as an overlay network, executing on some hosts and/or network nodes. The filter specifications from each subscriber are propagated towards the event producers, and at each internal node new filter specifications are merged with old ones. Event notifications which do not match any subscriptions are stopped early. The event notification service should also provide an API for notifying producers when there are no interested consumers, to improve scalability even more.

Filtering also allows a number of event consumers to share a single event producer in a conform way. As an example, consider a component specifying the filter given previously, and a new consumer on the same host which subscribes with a similar filter, where $motion > 0.3$. The event notification service resolves the filter specification incompatibility, but a more restrictive filter is effective immediately (although delivery is not optimized) while a less restrictive filter may have to propagate through all the event notification service nodes and possibly also notify the producer component. A low and predictable delay is important for the domain of real-time media processing, because it determines on which time scale filters may be used as a means of turning on and off event producers.

3.6. Event notification delivery

The event notification service must balance requirements for real-time communication, low event propagation delay, and high throughput against reliability and ordering guarantees associated with event notification delivery. Performance numbers for an implementation of the CORBA Notification Service [13] is given in [19] for both "best effort" delivery and the highest event delivery guarantee possible with CORBA notification service - consumers receive all events in spite of supplier, consumer, and notification service crashes and network failures. A decrease in performance of more than 80% is reported when using both event and connection persistence compared to "best effort" delivery. In both cases the number of event/second as seen by each consumer decreases rapidly as the number of consumers increases, because the event notification service itself performs the group communication.

Our approach is to handle the unreliability of event delivery in the same way as we handle the unreliability of analysis algorithms, which may fail to detect and report an event or report false positives. In our framework, the C components are designed to handle this by the use of a statistical approach [7]. Consequently, an event notification service which also supports unreliable transport, such as UDP, is desirable in order to capitalize on the scalability of native

IP multicast[1].

Another issue related to the real-time characteristics, is the buffering policy used in event notification delivery. The event notification service should provide an API for specifying a policy to use when buffers are filled up, such as "discard oldest" or "discard newest" event notification.

3.7. Event notification ordering

Applications built on top of our framework need to temporarily relate event notifications, potentially originating from arbitrarily distributed components, and hence some ordering mechanism is required. Each event notification is associated with a time interval, given by a *start* and a *stop* time, illustrated in Section 3.4. This indicates that we allow events to have a duration in time, supporting a wide class of sustained actions to be interpreted as events. A language for specifying temporal relations (e.g. *before*, *after*, and *within 10 seconds*) is needed, but not addressed in this paper. The aggregation of simple sustained events into composite sustained events is performed according to such a temporal specification. The ordering mechanisms must be sufficiently strong in order to support the specification language expressiveness. The design of our framework, supporting such global ordering, is based upon a common knowledge of time in all components.

It is well known that in a distributed system, global time can only be available down to a certain level of accuracy. The most widely used global time service in the Internet is NTP (Network Time Protocol, RFC 1305) where synchronization accuracies between different hosts is reported to be in the order of tens of milliseconds on the Internet in general and one millisecond over LANs[11]. In general the required level of accuracy is application dependent, but considering video streams with 25 frames per second, NTP may provide frame level accuracy even over wide area networks. Some additional inaccuracy is introduced by the indeterministic skew (OS scheduling, etc.) on the data path from capture device to the timestamping application. Hence, timestamping should happen as close to the source as possible. As an example, a computer with a video camera connected through a video grabber card, should timestamp each video frame in the video grabber card or in the driver. Inaccuracies introduced by NTP and the timestamping process introduce intervals of uncertainties, where the system is unable to determine ordering (e.g. which event took place first).

As a result, our framework does not require any kind of ordering or synchronization support in the event notification service. Ordering is handled and implemented at the framework level. Assuming that timestamps are obtained from globally synchronized clocks, components are able to detect and handle delays introduced by the event notification service. In SIENA[2] (Scalable Internet Event Notification Architecture) a similar approach is used to detect and account for latency effects.

As a conclusion, event notification services designed to provide ordering guarantees may have problems related to scalability, and in this case they are unsuited for our application domain.

3.8. Event notification size

As an example of a "medium sized" event notification, consider an E component doing motion estimation, treating each frame as a number of blocks of, say $16x16$ pixels, calculating a motion vector and difference value for each block on the basis of two consecutive video frames. A $640x480$ pixel frame size results in 1200 motion vectors and difference values. Another example of space consuming events is compressed images, an important class of data in media processing applications. A component may perform skin classification on particular video frames, where the intensity in the resulting gray scale image represents the skin color probability. An event notification service which is able to handle such potential space consuming event notifications is beneficial.

However, for filtering reasons all these data should probably not *always* be embedded inside a single event notification. There is a tradeoff between filtering granularity and performance. Event consumers may have interest in only parts of a video frame. Event notifications which fit in a single link layer frame reduces fragmentation and reassembly costs in the communication protocol stack and at the same time allows relatively fine grained filtering.

3.9. Event notification service and media streaming

Until now we have focused on the communication between E and C components and argued that an event-based interaction model fits well. From a media processing point of view, it is also interesting to use the event notification service itself for interaction between F and E components, and even for streaming time-based media (e.g. video) to F components. The extension of event notification services for handling stream based interaction internally has been described in [3] and [18].

An event notification service which is capable of handling such "stream events" is attractive, especially when used in combination with filtering. A producer publishing video frames as event notifications may associate a type, name, and value tuple expressing the type of a particular video frame, such as an I (intra), P (predictive), or B (bidirectional) coded frame. Different media processing components may then subscribe and register interest in I frames only, or in only every 10th I frame (temporal division), depending on the expressiveness of the filter specification

language. A producer may send different areas of a video frame as different event notifications, providing consumers with the ability to express interest in certain areas of a video frame only (spatial division). An E component doing motion estimation may subscribe only to events which contain the boarder blocks of a video frame, and a C component may use these values to detect entry or exit from a camera view.

4. Prototype

In this section we describe a prototype of the framework with emphasis on the parts relevant for distributed event-based systems.

4.1. Event notification service

Different event notification service technologies, such as CORBA Event Service[14], CORBA Notification Service, and SIENA are available. We are at the time of writing in the process of starting to use SIENA and CORBA Notification Service. The event notification service in this prototype is based on Mbus[16], which we will now describe and discuss.

Mbus is a standard[17] currently being worked out by the IETF. Mbus is designed to support *coordination and control* between different application entities, corresponding roughly to components in our terminology. The standard defines message addressing, transport, security issues and message syntax for a lightweight message oriented infrastructure for ad-hoc composition of heterogeneous components. The authors of Mbus state that Mbus is not intended for use as a wide area conference control protocol, for security (conforming Mbus implementations must support both authentication and encryption[17]), scalability, message reliability and delay reasons[16]. In the following we evaluate the suitability of Mbus as an event notification service, with respect to the requirements discussed in Section 3.

Mbus supports binding of different causalities by using a "broadcasting" and filtering technique. All components participating in a specific Mbus session subscribe to an IP multicast address and in effect Mbus messages are "broadcasted" to the set of computers hosting components participating in this Mbus session. Currently, Mbus is implemented as a library which is linked into the applications. By linking the Mbus code into the application itself, the Mbus layer in each component sees all Mbus traffic and must filter messages. Mbus could have been implemented as a part of the operating system, pushing the filtering one step closer to the event notification producer. In this respect Mbus is rather suboptimal. As an optimization, several Mbus sessions may be started, using a set of IP multicast addresses. Each component participate in a subset of these Mbus sessions and messages are sent to different sessions based upon some predefined scheme.

Important for event notification selection and filtering is the addressing used in Mbus. The address of a component is specified when initializing the Mbus layer. In other words, selection and filtering is associated with the address given to a component, not by specifying filters. The Mbus header includes source and destination addresses, each a sequence of attribute-value pairs, of which exactly one pair is guaranteed to be unique (combination of process identifier, process demultiplexer and IP address). Each Mbus component receives messages addressed to any subset of its own address. A Mbus component is able to address a single, *"(id:7-1@129.240.64.28)"*, a subset, *"(media_type:video component_type:E)"*, or all, *"()"*, Mbus components by specifying an appropriate sequence of attribute-value pairs. As a result, bindings between components are implicit.

It should by now be evident that Mbus supports push-based interaction. Some higher level Mbus services are described in [9], such as abstractions for remote procedure call. Pull style interaction is achieved by either sending requests as event notifications or by using the remote procedure call service.

An Mbus-based event notification service acts as a layer of indirection between components giving both access and location transparency, simplifying reconfiguration and migration. Component awareness is supported by a soft state approach, where the Mbus layer listens and periodically sends self-announcements messages on behalf of its component. When migrating a component to another host, its Mbus address remains the same (except for the value of the *id* attribute, reported in succeeding self-announcements).

Regarding scalability, message propagation delay, and reliability of event delivery, an Mbus-based event notification service inherits many of its characteristics from IP multicast, which is realized as a distributed and scalable service. The state necessary for forwarding IP multicast packets is calculated and stored in both routers and in hosts acting on behalf of multicast receivers in a distributed fashion. The Mbus component awareness functionality limits scalability, but the rate of self-announcements is adapted to the number of entities participating in a session.

IP multicast also decreases latency (by sending only one instance of a packet over any link) which is very important for the domain initially targeted by IP multicast, real-time, high bandwidth multi-user applications, such as video and audio conferences.

At the transport level, Mbus messages are encapsulated in UDP packets and transported unreliably by IP multicast. In the special case where the message is targeted at exactly one receiver, reliable unicast delivery is supported by the Mbus layer, using acknowledgement, timeout, and retrans-

missions mechanisms. The Mbus/UDP/IP multicast protocol stack does not give any ordering guarantees, but assuming global time and associating a time interval to each event (see section 3.7) handles this ordering problem, except for very time-sensitive applications.

From the discussion above, we believe that Mbus is a reasonable alternative as an event notification service for small scale experiments. From a prototyping viewpoint it is easy to integrate, requiring few lines of code, and the text-based message format simplifies message snooping.

4.2. F, E, and C, components

In the current prototype of the framework, ways to implement F, E, and C components have been identified.

We have used applications such as JMStudio, bundled with JMF[21] (Java Media Framework), and vic[10] for streaming video from a file or from a capture card connected to a video camera.

The F and E components are realized using JMF. JMF performs low level media tasks, such as capture, transport, streaming, (de)multiplexing, (de)coding, and rendering. JMF also provides a pluggable architecture for integrating custom media processing algorithms. The F and E components developed for the prototype are implemented as classes in the Java programming language and pluggable into the JMF framework. F and E components implement a method (e.g. iterating through all pixels of a video frame performing some calculations), which is invoked by the JMF system whenever a new media sample is available.

The C component implemented for this prototype is based on dynamic object-oriented Bayesian networks (a generalization of the hidden Markov model) and particle filters [7].

4.3. Component interaction

In our implementation of the media processing framework, the communication between media sources and F components is done by standard techniques for streaming media data, such as MPEG/RTP[20]/UDP/IP multicast. An E component executes, together with some F components, inside a single process and communication between these F and E components is handled by shared buffers and performed by JMF. Interaction between E and C components is handled by the Mbus-based event notification service.

4.4. Event notification ordering

As described in Section 3.7, a common knowledge of global time in all components is assumed. When media (e.g. video) is streamed by using RTP, each RTP packet contains an RTP timestamp. This RTP timestamp is only relative and it is used by receivers to determine the duration between two consecutive media samples. However it is possible for receivers to determine the global NTP time of the media sample in spite that such RTP packets contain no NTP timestamp themselves. RTP has a companion protocol, RTCP[20], which is used for sending reports about the session itself. Such reports are sent "out of band" on a separate port. RTCP packets include so called "sender reports" which, for each source gives a NTP timestamp and the corresponding RTP timestamp. When a receiver, a F component, has seen two such RTCP "sender reports" from a media source, it is able to derive a mapping from an arbitrary RTP time, for this source, to global NTP time. The timestamps then follow the filtered and transformed media data to E components, which inserts the timestamps into event notifications. Then C components use these timestamps in order to relate events temporaly.

4.5. Experiences

The purpose of the prototype is to gain experience, serve as a proof of concept, and to verify the flexibility of Mbus with respect to the different requirements discussed in Section 3.

Component interaction uses push style communication. The media source applications push video streams over the network, using IP multicast. The video streams are received by processes executing the JMF runtime system and hosting F and E components. In each process, the JMF runtime system invokes the media processing methods of F and E components whenever a new video frame arrives. Each E component pushes the calculated results, features extracted from video frames, over the event notification service. A C component, hosted by a separate process, receives these results and performs classification.

The deployment of components onto hosts is performed manually in this prototype. The flexibility of the Mbus-based event notification service was confirmed by experiments - configuration and compilation was unnecessary before starting components on different hosts or when reconfiguring the media processing task. Mbus performs as expected, both as an intra host event notification service, but also when components are hosted by different computers interconnected by a local area network. Some successful migration experiments have also been conducted [15], using Voyager[6] for moving running components between hosts.

We have performed some preliminary experiments, testing different distribution strategies. On one extreme, all components executed on a single machine while on the other extreme each component was hosted by a different computer. Distributed processing of the media anlysis task allows the development of more resource demanding, but also more reliable components, improving scalability.

5. Conclusions and future work

In this paper we have presented an architecture for a framework for developing applications supporting distributed real-time processing of time-based media streams. We have described different aspects of our target application domain and argued that this domain is a case for distributed event-based interaction. The main contribution of this paper is the analysis of different aspects relevant to the application domain and the translation to corresponding requirements for a suitable event notification service. These requirements include scalability, reconfiguration, migration, event notification selection and filtering. Ordering is handled at the framework level by assuming globally synchronized clocks and association of timestamps to events. A prototype of the framework has been implemented, serving as a proof of concept and for evaluation purposes. We find the results promising.

We are currently working on a new prototype of both framework and test application. The test application chosen is tracking of objects in video, an application which is challenging and extensible along several axes. It should be possible to increase complexity, with respect to feature extraction and classification, by going from object tracking to recognition of persons, and maybe also their identity. For scalability tests, a number of video streams can be analyzed concurrently. Additionally, some of the algorithms are relatively compute intensive, but also suited for parallel processing (e.g. motion estimation).

6. Acknowledgments

We would like to thank all persons involved in the Distributed Media Journaling project for contributing to ideas presented in this paper. We also would like to thank the reviewers for valuable comments.

The DMJ project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431.

References

[1] K. C. Almeroth. The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment. *IEEE Network*, 2000.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, Portland OR, USA, July 2000.

[3] D. Chambers, G. Lyons, and J. Duggan. Stream Enhancements for the CORBA Event Service. In *Proceedings of the ACM Multimedia (SIGMM) Conference, Ottawa*, October 2001.

[4] DMJ, (Distributed Media Journaling) project. http://www.ifi.uio.no/˜dmj/.

[5] V. S. W. Eide, F. Eliassen, and O. Lysne. Supporting Distributed Processing of Time-based Media Streams. In *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA 2001), Rome, Italy*, pages 281–288, Sept 2001.

[6] G. Glass. Voyager - the universal orb. Technical report, Objectspace, January 1999.

[7] O.-C. Granmo and F. V. Jensen. Real-time Hyphothesis Driven Feature Extraction on Parallel Processing Architectures. In *Proceedings of the 2002 Special Session on Parallel and Distributed Multimedia Processing & Retrieval (PDMPR 2002), Las Vegas, USA*, June 2002.

[8] S. Hongeng, F. Brémond, and R. Nevatia. Bayesian Framework for Video Surveillance Application. In *Proceedings of the 15th International Conference on Pattern Recognition*, pages 164–170, Sep 2000.

[9] D. Kutscher. The Message Bus: Guidelines for Application Profile Writers. *Internet Draft , draft-ietf-mmusic-mbus-guidelines-00.txt*, 2001.

[10] S. McCanne and V. Jacobsen. Vic: A flexible Framework for Packet Video. *In ACM Multimedia'95, pp. 511-522*, 1995.

[11] D. L. Mills. Improved Algorithms for Synchronizing Computer Network Clocks. *IEEE Transactions Networks*, pages 245–254, 1995.

[12] Y. Nakamura and M. Nagao. Parallel Feature Extraction System With Multi-Agents-PAFE. *11th IAPR International Conference on Pattern Recognition (ICPR)*, vol. 2:371–375, 1992.

[13] Object Management Group Inc. CORBA services, Notification Service Specification, v1.0. *http://www.omg.org/*, 2000.

[14] Object Management Group Inc. CORBA services, Event Service Specification, v1.1. *http://www.omg.org/*, 2001.

[15] R. W. Olsen. Component Framework for Distributed Media Journaling. Master's thesis, (in Norwegian), Department of Informatics, University of Oslo, May 2001.

[16] J. Ott, D. Kutscher, and C. Perkins. The Message Bus: A Platform for Component-based Conferencing Applications. *CSCW2000, workshop on Component-Based Groupware*, 2000.

[17] J. Ott, C. Perkins, and D. Kutscher. A message bus for local coordination. *Internet Draft , draft-ietf-mmusic-mbus-04.txt*, 2001.

[18] T. Qian and R. Campbell. 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, May 1997.

[19] S. Ramani, B. Dasarathy, and K. S. Trivedi. Reliable Messaging Using the CORBA Notification Service. In *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA 2001), Rome, Italy*, pages 229–238, Sept 2001.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobsen. RTP: A Transport Protocol for Real-Time Applications. *RFC 1889*, 1996.

[21] Sun Microsystems Inc. Java Media Framework, API Guide, v2.0. *http://java.sun.com/*, 1999.

Paper III

Scalable Independent Multi-level Distribution in Multimedia Content Analysis

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

Published: In Proceedings of Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS 2002), volume 2515 of Lecture Notes in Computer Science, Springer, pages 37-48, Coimbra, Portugal, November 2002.

Evaluation: In total, 112 papers were submitted to IDMS/PROMS 2002. The review work was done by the program committee members and additional reviewers. As a result, 30 papers were accepted for publication.

Author Contribution: This is a project article, where Granmo and Eide did most of the writing. Granmo is the sole author of section 3.5, while sections 3.1 - 3.4 were written by Eide. The remaining sections were written iteratively, mainly by Granmo and Eide. The supervisors have contributed to the ideas presented in this paper and commented on different draft versions of the paper after each iteration. In order to integrate the results, the experiments were designed collaboratively. Granmo implemented the classifier part, while Eide did the rest of the implementation.

[1] Authors are listed alphabetically

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

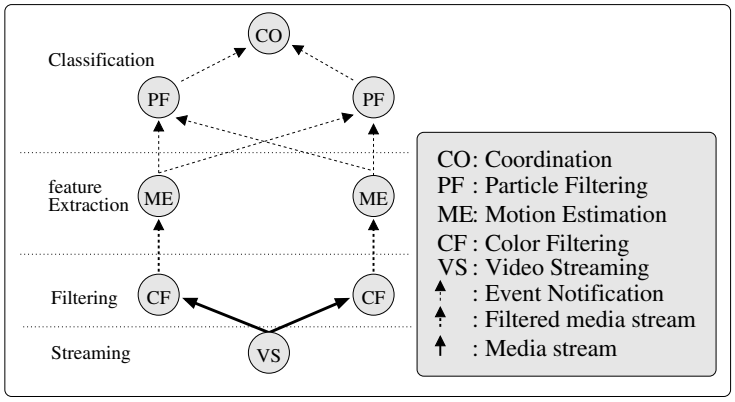

Fig. 1. A specific configuration, out of many possible configurations, of a content analysis application for real-time tracking of a moving object in a video stream

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.

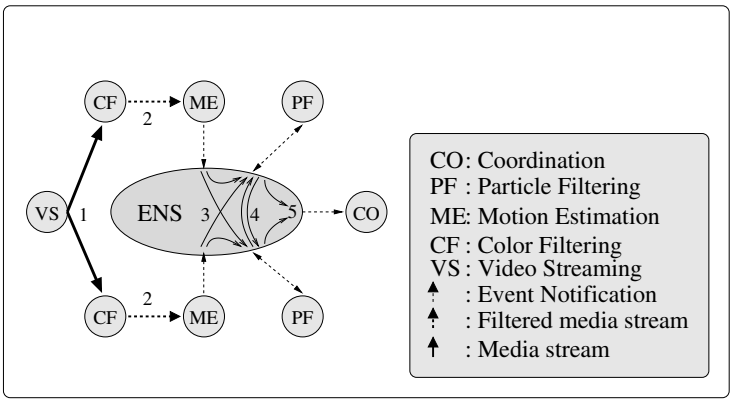

Fig. 2. Inter component communication for the configuration in Fig.1. Feature extraction and classification components interact through an Event Notification Service, labeled ENS

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

Paper IV

Extending Content-based Publish/Subscribe Systems with Multicast Support

Viktor S. Wold Eide, Frank Eliassen,
Olav Lysne, and Ole-Christoffer Granmo

Published: Technical Report 2003-03, Simula Research Laboratory, July 2003.

Author Contribution: The work presented in this technical report was primarily conducted by Eide, i.e., the work related to the requirement analysis, specification, design, implementation, validation, experimentation, and the paper writing. The contributions of he coauthors are commenting on draft versions of the report.

Extending Content-based Publish/Subscribe Systems with Multicast Support

Viktor S. Wold Eide[1,2], Frank Eliassen[2], Olav Lysne[2], and Ole-Christoffer Granmo[1,3]

[1]University of Oslo
P.O. Box 1080 Blindern
N-0314 Oslo, Norway
viktore,olegr@ifi.uio.no

[2]Simula Research Laboratory
P.O. Box 134
N-1325 Lysaker, Norway
viktore,frank,olavly@simula.no

[3]Agder University College
Grooseveien 36
N-4876 Grimstad, Norway
ole.granmo@hia.no

Simula Research Laboratory
Technical Report 2003-03

July 2003

Abstract

Event-based interaction is recognized as being well suited for loosely coupled distributed applications. Current distributed content-based event notifications services are often architectured to operate over WANs (wide area networks). Additionally, one to one transport layer communication primitives are used. As a result, these services are not suitable for (parts of) applications having a combination of high event notification rates and locally a large number of interested parties for the same event notifications.

In this paper we describe the architecture of a distributed content-based event notification service, designed to take advantage of the available performance and native multicast support provided by current "off the shelf" network equipment. Our event notification service is designed primarily as a LAN (local area network) service and hence complementary to event notification services for WANs. A prototype has been implemented and experiments indicate that our service provides both scalability and high performance. A client may publish several thousand event notifications, carrying several MBytes of data, per second. The service is unaffected by the number of interested parties, due to the use of native multicast.

1 Introduction

Traditionally, the client/server interaction model has been used extensively for building distributed applications. However, a large class of distributed applications are better structured as a number of asynchronously processing and communicating entities. Such applications fit well to the publish/subscribe interaction paradigm, leading to an event-based interaction model. Event-based interaction provides a number of distinguishing characteristics, such as asynchronous many to many communication, lack of explicit addressing, indirect communication, and loose coupling.

Event-based systems rely on some kind of event notification service, as illustrated in Figure 1. A distributed event notification service is realized by a number of cooperating servers, also denoted *brokers* in the literature. Clients connect to these servers and are either *objects of interest*, *interested parties*, or both. An object of interest publishes event notifications, or just notifications for short. Some systems may allow/require the object of interest to advertise the notifications potentially generated before publishing. Interested parties subscribe in order to express interest in particular notifications. The responsibility of the event notification service is routing and forwarding of notifications from objects of interest to interested parties. In essence, the servers jointly form an overlay network of notification routers. A survey of the publish/subscribe communication paradigm and the relations to other interaction paradigms are described in e.g. [6].

Publish/subscribe systems differ with respect to the expressiveness of their subscription languages. In *channel-based* systems, e.g. as specified by the CORBA Event Service [8], an interested party may subscribe to a channel and in effect receive all notifications sent across that particular channel. *Subject-based* systems, such as e.g. TIBCO Rendezvous[14], provide somewhat finer granularity with respect to selection of notifications. An object of interest determines the most appropriate subject for each notification published. The content of a notification is not used by the service for forwarding. Subject-based systems may also support hierarchical subject names and/or wild-card expressions on subject identifiers to further improve the expressiveness of subscriptions. *Content-based* publish-subscribe systems, such as Elvin[12], Gryphon[9], Hermes[11], and SIENA[2] provide even finer granularity. In such systems notifications typically consist of a number of attribute/value pairs. A subscription may include an arbitrary number of attribute names and filtering criteria on their values. Hence, content-based systems increase subscription selectivity by allowing subscriptions along multiple dimensions.

Distributed content-based publish/subscribe systems, such as Gryphon, Hermes, and SIENA, are often architectured to operate over WANs, e.g. public networks or the Internet. A main concern is how to efficiently distribute event notifications between servers. E.g. in [9], the servers are treated as the communication endpoints.

In contrast, in this report we are mainly concerned about how to efficiently distribute very high rate event notifications between a large number of objects of interest and interested parties within a smaller region, e.g. a LAN or an administrative domain. A scalable and high performance event notification service allows development of new classes of applications which utilize event-based interaction, e.g. high performance parallel computing within clusters of powerful computers and real-time video streaming to clients hosted by heterogeneous computers and network connections. The application domain of real-time content analysis[4] covers both these areas. A service capable of handling the data rates of several concurrent high quality real-time video streams is definitely useful for other application domains as well. Highly important is how to transport the notifications all the way from the objects of interest and to the clients, i.e. not only between the servers.

With respect to the communication path between an object of interest and a server, it is important to ensure that only the relevant notifications are generated and sent, i.e. to support filtering at the source. Elvin relies on a quenching mechanism[12] where (parts of) the subscription database is sent to objects of interest. This strategy is described as relatively complex and is optional in order to support thin clients. With respect to the communication path between a server and the interested parties, efficient multicast is crucial in order to distribute each notification to a large number of interested parties. A notification sent by *native* multicast requires only a single send operation and propagates over each network link only once, regardless of the number of computers hosting interested parties and the number of interested parties hosted by each computer.

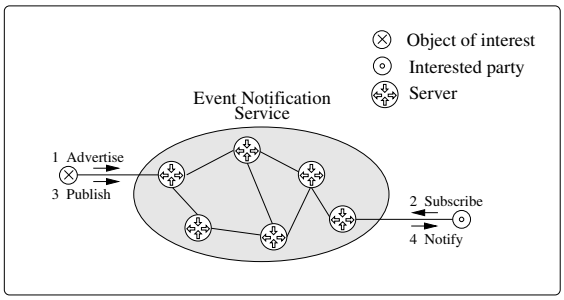

Figure 1: A Distributed Event Notification Service

Utilizing native multicast communication in channel-based and subject-based publish/subscribe systems is relatively straightforward. In such systems, a notification is basically mapped onto a channel or a subject which is then mapped onto a multicast address.

The principles and techniques used for routing and forwarding notifications between servers in distributed content-based publish/subscribe systems are similar to those used by IP routers in order to support IP multicast. In e.g.[1], an efficient multicast protocol for content-based publish/subscribe systems is described. The challenge of utilizing *native* multicast support for content-based publish/subscribe systems is well known[6]. Simulation results for some algorithms for distributing notifications in a network of brokers is presented in [9]. But to our knowledge, native multicast support has not been implemented in current content-based publish/subscribe systems. Hence, it may be desirable to enhance existing content-based publish/subscribe systems to take advantage of network level multicast communication. A major challenge is how to achieve this while affecting neither the API nor the semantics of the event notification service.

Some event notification services adopt a hierarchical approach where the intra domain and the inter domain cases are handled differently, e.g. [14], using specialized routing daemons between domains. A hierarchical approach is particularly useful when notifications have high regionalism, i.e. when notifications have high interest in certain parts of the network and little or no interest in other parts. Non-uniform distribution of subscriptions is likely due to e.g. distributed applications built with locality in mind for performance and cost reasons, location based services, security, what people are interested in, etc.

In this paper we describe the architecture, the implementation, and the measured performance for our distributed content-based event notification service. The service takes advantage of the native multicast support provided by current network equipment, such as switches and network interface cards. By limiting ourselves to the LAN/administrative domain case, we are able to make certain assumptions not acceptable in the WAN case. We envisage that instances of our event notification service software are executed inside LANs, but connect to a WAN event notification service, by e.g. gateways/routing daemons. We therefore view our work as complementary to WAN event notification services.

In our current implementation so-called mapping specifications are generated manually, but may be changed during runtime. The principle used to determine such specifications is to map notifications generated at a high rate onto separate multicast channels, i.e. to isolate such high rate traffic. Our approach is useful for a large class of applications and also a natural first step towards a more dynamic solution, where mappings are generated and updated automatically during runtime.

The rest of the report is structured as follows. First we provide some background information in Section 2. Then we present the requirements for our event notification service in Section 3. Based on these requirements, we describe the architecture of our service in Section 4. In Section 5 we describe our prototype. In Section 6 we describe some experiments and present some empirical results. In Section 7 we discuss some ideas for further work. Lastly, in Section 8 we conclude.

Method
$publish(notification\ n)$
$subscribe(string\ identity,\ pattern\ expression)$
$unsubscribe(string\ identity,\ pattern\ expression)$
$advertise(string\ identity,\ filter\ expression)$
$unadvertise(string\ identity,\ filter\ expression)$

Table 1: Interface of SIENA

2 Background

In this section we provide some more background information for content-based publish subscribe systems. Our description is biased towards SIENA[2], on which we have based our prototype implementation.

2.1 Event Notification Service API

An event notification service typically provides a method used by objects of interest to publish notifications and a method used by interested parties to register interest in notifications. Additionally, the event notification service may provide a method used by an object of interest to inform the event notification service about the kind of notifications potentially generated. Methods for unregistering are typically also available. As an example, Table 1 illustrates the interface of SIENA. Objects of interest and interested parties must identify themselves to the event notification service, which maintains references to the clients. A pattern is basically a sequence of filters[2], but in the rest of the paper we assume that filters are used for subscriptions.

2.2 Event Notifications

In SIENA, an event notification is basically a set of *type, name, value* tuples. The most common types are supported, e.g. string, integer, float, etc.

An example of a notification is given in Table 2. This particular notification contains a small region of a video frame - the luminance part, encoded in 8 bits of resolution. The video frame region is a rectangular block, $16x16$ pixels. The encoding is represented as a string, for illustration purposes. This block is intra coded, i.e. independent from blocks in earlier and later frames. The *pixels* attribute contains the pixel values for this $16x16$ block, illustrated as characters. This particular block has a $(horizontal, vertical)$ placement within the frame of $(1, 5)$. Video client software may translate spatial and temporal requirements into subscriptions, based on the particular encoding scheme used.

In this report we have used real-time video streaming as an example of an application domain requiring a high performance event notification service. The challenge of supporting heterogeneous receivers in video streaming applications is well known. As an example, a combination of a layered video compression algorithm and receiver driven multicast is described in [7], providing scalable multicast video transmission. Each layer encodes a portion of the video signal and is sent to a designated IP multicast address.

In an event-based approach, each video frame may be published as a number of notifications. E.g. in [3], an extension for the CORBA Event Service is described which supports stream events and multicast delivery. Content-based publish/subscribe systems have the potential of supporting even more fine grained selection, compared to direct use of multicast or a channel-based approach. Interested parties may subscribe to only a certain part of a video stream as explained above and thereby reduce resolution both spatially and temporally. Additionally, a content-based approach may better support parallel processing, as pointed out in [4].

Type	Name	Value
string	*media_type*	*video*
string	*media_source*	*fnasa.simula.no*
string	*encoding*	*/raw/luminance*/8/16*x*16
string	*block_type*	*intra_coded*
integer	*block_h*	1
integer	*block_v*	5
byte[ ]	*pixels*	*q*34*i*23*QR ... D*

Table 2: Event Notification Example

2.3 Filters

A filter is a sequence of attributes and constraints on the attributes. The constraints are specified in a constraint language, which also defines some supported operators. The expressiveness of such languages may differ, but an important design issue is the balance between the expressiveness of the language and the associated computational complexity[2]. A filter may be used in different contexts - for subscriptions or advertisements.

An example of a filter for subscriptions is given in Table 3. This filter may be used to express interest in notifications from a particular source, which contain video data, with a particular encoding, with a particular block type, but only the 10 leftmost (block) columns.

An example of a filter for advertisements is given in Table 4. This filter may be used to advertise notifications which will contain parts of a particular encoded video stream, but only the intra encoded blocks and only the two leftmost columns, where *block_h* is 0 or 1.

2.4 Filtering

In very simple publish/subscribe systems, each server may broadcast notifications to all other servers. All servers connect to a well known multicast address and notifications are sent to the multicast address and are then forwarded by the *multicast service*. Each server then filters notifications on behalf of their interested parties. The result of this late filtering is reduced scalability, as both network bandwidth and processing resources are wasted. This is the approach used in Mbus[10]. Mbus is designed to support *coordination and control* between different application entities hosted by different computers on a LAN. In [5] we have evaluated the suitability of Mbus as a LAN event notification service.

For most applications the number of notifications is likely to be significantly larger than the number of subscriptions. Hence a better approach is to broadcast subscriptions, which are then used to prune the delivery of notifications. The gain of this strategy clearly depends on the ratio of notifications to subscriptions.

Similarly, in publish/subscribe systems which support advertisements, the advertisements may be broadcast and used to prune the delivery of subscriptions, which are then used to reduce the flow of notifications.

It should be noted that in the last two approaches, subscriptions/advertisements are not forwarded unless they are more general than the current forwarded ones.

In Hermes[11], a different approach is used. So-called rendezvous nodes ensure that subscriptions and advertisements meet somewhere between objects of interest and interested parties, without any global broadcasts.

2.5 The Covering Relation

The covering relation is described in [2] and is important in order to understand the rest of the report. The relation $x \prec_Y^X y$ is read as x *matches* y or alternatively y *covers* x. X and Y indicate the type of x and y respectively and may be of type N (Notification), S (Subscription filter), or

Type	Name	Value/Expression
string	*media_type*	*video*
string	*media_source*	$fnasa.simula.no$
string	*encoding*	$/raw/luminance/8/16x16$
string	*block_type*	*intra_coded*
integer	*block_h*	$>= 0$
integer	*block_h*	< 10

Table 3: Subscription Filter Example

Type	Name	Value/Expression
string	*media_type*	*video*
string	*media_source*	$fnasa.simula.no$
string	*encoding*	$/raw/luminance/8/16x16$
string	*block_type*	*intra_coded*
integer	*block_h*	$>= 0\ AND\ < 2$
integer	*block_v*	*ANY*
byte[ ]	*pixels*	*ANY*

Table 4: Advertisement Filter Example

A (Advertisement filter). In the following, the symbol α represents an attribute in a notification and the symbol ϕ represents an attribute constraint in a subscription or advertisement filter. The most important relations are:

$\alpha \prec \phi \Leftrightarrow type_\alpha = type_\phi \wedge name_\alpha = name_\phi \wedge operator_\phi(value_\alpha, value_\phi)$: The attribute α matches the attribute constraint ϕ if and only if the types and names are identical and the operator returns true

$n \prec_S^N s \Leftrightarrow \forall \phi \in s : \exists \alpha \in n : \alpha \prec \phi$: : The notification n matches the subscription filter s if and only if each and every attribute constraint in s is matched by an attribute in n. Multiple constraints for the same attribute is interpreted as a conjunction

$n \prec_A^N a \Leftrightarrow \forall \alpha \in n : \exists \phi \in a : \alpha \prec \phi$: The notification n matches the advertisement filter a if and only if each attribute in n is matched by an attribute constraint in a. Multiple constraints for the same attribute is interpreted as a disjunction

$s \prec_A^S a \Leftrightarrow \exists n^N : n \prec a \wedge n \prec s$: The subscription filter s matches the advertisement filter a if and only if there exists a notification n which matches both s and a. In other words, if the set of notifications defined by s and the set of notifications defined by a have nonempty intersection

2.6 The Mapping Problem

In general, each notification is of interest to a subset of all interested parties. Theoretically and ideally, a multicast address may be used for each possible combination of computers hosting the interested parties. In practice this is not possible, as the required number of multicast addresses grows exponentially and quickly beyond practical limits.

In [9], some algorithms are presented, targeting this mapping problem. In the article, brokers, and not the computers hosting clients, are treated as the communication endpoints. In addition to the theoretically ideal algorithm, five algorithms are presented. The principles used in order to reduce the required number of multicast groups are to *reduce group precision* (brokers receive and filter out notifications which are of no interest to its clients), *send multiple multicast*, and

send over multiple hops. In all algorithms, except a so-called *group approximation algorithm*, the mapping is static. Simulation results on a wide area network are presented. The authors find that a flooding approach is viable over a range of conditions, but in case of high selectivity and high regionalism of subscriptions the non flooding approaches are significantly better.

3 Service Requirements

In this section we describe the requirements for our scalable and high performance LAN event notification service.

3.1 Exploit Locality

The programming of applications utilizing event-based interaction should to a reasonable extent be handled independently from the deployment, i.e. where clients are instantiated and where they are executed. A different API should not be necessary, e.g. when an object of interest and an interested party for performance reasons are deployed on the same computer. Therefore, our event notification service must *efficiently* support:

- *intra LAN communication*: between objects of interest and interested parties hosted by different computers connected via a LAN
- *intra host communication*: between objects of interest and interested parties hosted by different processes on the same computer
- *intra process communication*: between objects of interest and interested parties hosted by a single process

The first case is important for distributed applications executing on a LAN. Although LANs offer vast amounts of bandwidth and short delay, this is not automatically the case for an event notification service deployed on top of it. Care must be taken in order to provide performance close to the bare hardware capabilities. A high performance service may allow applications within the domain of high performance parallel computing to utilize event-based interaction for efficient communication, e.g. between powerful computers within a cluster or on a LAN.

By supporting the second case, it becomes easier to take advantage of the processing capabilities of multiprocessor computers. Additionally, different processes and hence address spaces provide protection, both for a single user and between different users. It also allows application development by using a single computer.

Considering the third case, if notifications published by an object of interest are not of interest to any clients outside the process itself, then no such notifications should ever leave the process. Having both objects of interest and interested parties inside the same process is useful in order to exploit locality, e.g. to avoid copying large amounts of data between different address spaces. The exchange of notifications in this case must happen directly, i.e. without relying on some other process hosted by the same or another computer.

3.2 Utilize Multicast

The event notification service must also be able to take advantage of native multicast support in order to reduce the demand for both *processing* and *network* resources.

By utilizing multicast, only a single send operation is required by a server in order to publish a notification. In effect, the processing requirements are independent of the number of other computers hosting interested parties and the number of interested parties hosted by each computer.

With respect to network resources, the benefit of using multicast depends on both the applications and the underlying LAN technology. For LAN technologies which are broadcast by nature (e.g. wireless) or by design (e.g. traditional Ethernet), the cost of sending a single packet is basically the same for multicast and unicast. If the event notification service uses one to one transport

layer communication, each notification in effect is broadcasted several times, i.e. *all* computers on the LAN receive the same notification several times, dramatically reducing the performance. For switched wired LANs, supporting multicast natively, the situation is somewhat similar. If the event notification service is incapable of utilizing native multicast and there are several computers hosting interested parties, each notification will propagate over some links several times.

It is important to consider the mapping of network layer multicast onto link layer multicast, because this mapping is not always one to one. As an example, several IP multicast addresses could map to the same Ethernet multicast address.

An event notification service utilizing IP multicast inherits its dynamic properties. An IP multicast address is dynamically associated to a group of computers, by means of protocols such as IGMP (Internet Group Management Protocol). As an example, consider the case where a server is hosted by a computer for which the IP address is changed. As long as the computer continues to register interest in the multicast addresses, other servers hosted by other computers do not need to be aware of this change. As a result, IP multicast may also simplify runtime reconfiguration.

3.3 Support Runtime Reconfiguration

We expect that a large class of applications built on top of an event notification service may have rather dynamic characteristics, but along different dimensions. When considering an event notification service architecture, it is important to distinguish between changes in: the *number of objects of interest*, the *number of interested parties*, the *location* of clients, the *notification types* used by objects of interest, the notification *publishing rates*, and the *subscriptions* made by interested parties. Therefore, in order to adapt to the communication pattern of the applications, it should be possible to change the way notifications are mapped onto multicast addresses during runtime, without affecting the semantics of the service. Such reconfigurations should happen quickly and the performance should remain close to normal during such periods.

3.4 Provide Robustness

A distributed event notification service should provide robustness and tolerate certain failures. As an example, process, operating system, host, and link failures must not render the whole service useless. A link failure may partition the LAN into groups of computers which are not able to communicate with each other. However, the event notification service should still continue to operate inside such partitions. The value of the service should degrade gracefully.

4 Architecture

Based on these requirements, we now describe the architecture of our event notification service. First we discuss some assumptions which are typically acceptable for LANs, but not always for WANs.

4.1 Assumptions

For LANs it is reasonable to assume a single administrative domain. It is also reasonable to assume that IP multicast is (made) available within a domain and that the network equipment supports multicast natively. The mapping between notifications and IP multicast addresses may be done locally, within the domain, and administratively scoped IP multicast addresses may be used (RFC2365). In other words, the mapping is invisible outside the administrative domain.

The number of computers inside a LAN is also relatively limited, which is important when considering both architecture and algorithms. Additionally, stationary computers within a LAN often have relatively large amounts of computational capacity.

For wired equipment inside a LAN it is reasonable to assume low latency, typically sub millisecond, and lots of bandwidth, typically 100Mbps - 1 Gbps switches and network interface cards. Additionally, both jitter and the risk of packet loss are likely to be low.

4.2 Exploit Locality

A single server may be sufficient within a LAN as long as the number of objects of interest, the number of interested parties, and the publication rates are low. In this case, it may even be considered reasonable that notifications for which there are no interested parties are filtered at the server side. However, as the publishing rate increases and the number of objects of interest increases a quenching mechanism becomes important.

Similarly, as the number of clients which have interest in the same notifications increases, a server which handle each client separately by utilizing unicast will run out of steam.

By replacing the single server with a number of servers the processing is distributed between the servers, but the total network bandwidth consumption will most likely increase.

A native multicast approach is required in order to reduce this bandwidth consumption problem and to distribute notifications to a potentially very large number of interested parties. But in order to utilize native multicast, the computers hosting clients also must execute some software in order to determine which multicast addresses to subscribe to.

This reasoning indicates that each computer which hosts clients also should execute some software in order to handle quenching and subscriptions to IP multicast addresses. Therefore, in our architecture each computer hosting clients execute part of the event notification service, i.e. the software responsible for the intra process, the intra host, and the intra LAN event notification service. As a result, the event notification service software is executed cooperatively by computers within a LAN, which are often fairly powerful. However, some computers may act as *dedicated servers*, i.e. hosting no clients, or *thin clients*, i.e. interacting with the service through a server on another computer, but in this report we will not discuss these cases any further.

The software for the intra process case provides clients with the event notification service API. The intra host software is responsible for aggregating subscriptions for all the clients on the host as well as for executing the LAN event notification service protocol. In which context the software is actually executed on a particular computer is an implementation and deployment issue which may be realized in different ways, e.g. within a client process, within a separate process, within the operating system, or combinations of these. As an example, the intra process software may be implemented as a library, while a possible implementation of the intra host software may be a daemon process which is started whenever the first client is instantiated on a particular computer. In this case, the daemons hosted by different computers within the LAN exchange information and cooperatively realize the distributed event notification service.

In the following we will continue to use the term "server" and generally assume that all computers which host clients also host servers.

In our current approach, each server informs the other servers about the most general subscriptions made by their locally interested parties, i.e. subscriptions are used in order to prune the delivery of notifications.

4.3 Utilize Multicast

In order to take advantage of multicast, the challenge of mapping event-based communication onto multicast communication must be addressed. In the following we discuss our approach and issues related to this mapping problem. Note that we plan to extend our event notification service in order to take advantage of different transport protocols concurrently, but in this paper the emphasis is on utilizing native multicast support.

Additionally, note that the service described in this paper does not give any guarantees with respect to race conditions. As an example, interested parties may receive notifications even after the *unsubscribe*() method has been called. Clients are required to handle such cases. This is similar to the *best-effort* service as provided by SIENA[2].

Advert. Filter	→	Communication Identifier
f_1^A	→	$udp\_ipm : //239.0.10.1 : 6666$
f_2^A	→	$udp\_ipm : //239.0.10.2 : 6666$
f_3^A	→	$udp\_ipm : //239.0.10.3 : 6666$
f_4^A	→	$udp\_ipm : //239.0.10.4 : 6666$

Table 5: Mapping Specification Example

4.3.1 Mapping Specification

Table 5 gives an example of a mapping specification, where each row specifies an advertisement filter and a communication identifier. The communication identifier consists of a protocol name and a protocol specific part. The protocol used for all entries in this table is *udp_ipm*, our protocol for encapsulating notifications in UDP packets and transmission by IP multicast. The protocol specific part specifies different IP multicast addresses and a port number.

For now we assume that each server has a private copy of the mapping specification table. The table is required in order to handle subscriptions and publications.

First we describe how a subscription made by an interested party may make a server register interest in IP multicast addresses. Then we describe how a server maps notifications onto IP multicast addresses, which are then forwarded to the appropriate servers by the multicast service.

4.3.2 Subscriptions

When an interested party subscribes with the filter s^S as parameter, its server (executing on the same computer) checks if s^S is covered by subscriptions already made by any of its clients. If s^S is covered by current subscriptions, the server just register this interested party.

Otherwise the server must also make sure all the other servers become aware of this new subscription (e.g. by sending s^S on a well known multicast address, which all servers have registered interest in). Additionally, the server consults the mapping specification in order to determine which communication channels may potentially carry notifications covered by s^S. The table is checked sequentially. If f_j^A covers any notifications which are also covered by s^S, i.e. $s \prec_A^S f_j$, the server must make sure it will receive these notifications. As an example, if s^S is the subscription filter in Table 3 and f_2^A is the advertisement filter given in Table 4, then the server must register interest in the multicast address specified by $udp\_ipm : //239.0.10.2 : 6666$, since $s \prec_A^S f_2$. Observe that a subscription filter may cover (partially) *several* advertisement filters. In order to maintain the semantics of the service, the server therefore may have to register interest in several multicast addresses.

4.3.3 Publications

When an object of interest publishes a notification n^N, its server (executing on the same computer) checks if any subscriptions made by other servers cover n^N. If this is the case, the mapping specification is consulted in order to determine the associated communication identifier. The advertisement filters are checked sequentially. If n^N is covered by f_j^A, then the server sends n^N to the associated multicast address. The server also checks if there are any locally interested parties. If this is the case, these are also notified.

As an example, assume that n^N is the notification given in Table 2, s^S is the subscription filter given in Table 3, and f_2^A is the advertisement filter given in Table 4. If f_1^A does not cover n^N and f_2^A covers n^N and there is another server which has made the subscription s^S on behalf of its client(s), then n^N is sent to the multicast address specified for f_2^A, i.e. $udp\_ipm : //239.0.10.2 : 6666$.

Note that in our current architecture a notification is sent only once, on the multicast address associated with the first advertisement filter which covers the event notification.

4.3.4 Mapping Mismatch and Filtering

A notification is only forwarded by a server if it is covered by one or more subscriptions. If there is only a single interested party, only notifications covered by its subscriptions are forwarded. Filtering is performed early, by the server executing on behalf of an object of interest.

Depending on both the current mapping specification and the current subscriptions, some filtering may happen on the interested party side. If an interested party has specified a restrictive subscription filter and another interested party has specified a more general subscription filter and all notifications are mapped to the same multicast address, then the server executing on behalf of the first interested party must discard some notifications arriving on the multicast address.

The penalty for mapping mismatches is paid in terms of wasted network bandwidth and computational resources, raising the question of how to determine mapping specifications.

4.3.5 A Simple Mapping Heuristic

The following simple heuristic is currently used to generate mappings: Notifications generated at a high rate, of large size, and not of interest to all interested parties are mapped onto separate multicast addresses.

E.g., assume that a single mapping entry is used initially, which maps all possible notifications to a single multicast address. The effect of this mapping specification is late filtering, i.e. by servers hosting interested parties. As long as the rate of notifications is low or all interested parties have similar interests, this is most likely acceptable for a LAN service. For broadcast based LANs this is the effect anyway from a network point of view, although not from a processing or power consumption point of view.

However, a problem arises if one or more objects of interest start generating notifications at a very high rate (e.g. for publishing real-time video) and only some of the interested parties have subscribed to these notifications. Many servers would then have to discard these high rate event notifications, wasting network and processing resources.

Our approach allows a new entry to be installed into the mapping specification table, which maps all notifications part of a high rate notification "stream" to a different multicast address. When the mapping specification is updated, each server must check subscriptions made by its interested parties against the new mapping specification and register interest in the appropriate multicast addresses. Following the example above, the servers executing on behalf of clients interested in the high rate notifications (e.g. the video stream) would find their subscriptions to match the new advertisement filter and register interest in the newly announced multicast address. Similarly, the server executing on behalf of the publisher (e.g. the video server), would map and then send these high rate event notifications to this multicast address. As a result, the event notifications generated at a high rate are only forwarded to computers hosting interested parties which have actually subscribed to (some of) these notifications.

4.4 Runtime Reconfiguration

In the following we argue that some of the runtime changes discussed in Section 3.3 are handled by the IP multicast service while others should be handled by updating the mapping specification.

Due to the distributed architecture, changes in the *number of objects of interest* and the *number of interested parties* will have little effect on the service. The consumption of processing and network resources is distributed between the computers hosting clients. Additionally, by utilizing IP multicast, changes in the number of interested parties will have little impact on the service. A notification is sent at most once by a server anyway.

The dynamic properties of IP multicast also simplifies changes in client *location*. An object of interest which changes location may continue to publish notifications through a server. The computer hosting the server may send to IP multicast groups without joining them. An interested party which changes location will continue to receive notifications since its server registers interest in the appropriate IP multicast addresses and the computer joins IP multicast groups accordingly. Consequently, the number of clients and their location may change radically during runtime.

The *notification types* used by objects of interest are not likely to change very often. But by updating the mapping specification during runtime and hence reconfiguring the service, new types of notifications may get introduced and efficiently handled.

With respect to changes in notification *publishing rates*, some objects of interest may generate notifications with a relatively fixed rate while others may generate notifications sporadic. If it is possible to determine the parts of the "event notification space" *potentially* generated at a high rate, either a priori or during runtime, a mapping specification which partitions the "event notification space" accordingly may be used to reconfigure the service. For all notifications generated at a low rate, a single or only a few multicast addresses are sufficient since the client side filtering in this case is acceptable.

Clients which frequently subscribe and unsubscribe to the *same* notifications are handled similarly to changes in the number of interested parties. For interested parties which frequently change their *subscriptions* in order to receive a different part of the "event notification space", such changes are most likely limited to within part of the "event notification space". E.g., client software written in order to receive notifications carrying stock ticker information will not register interest in notifications carrying video data. By updating the mapping specifications, different parts of the "event notification space" may be further partitioned or merged in order to better match the subscriptions made by clients.

In effect, the mapping specification introduces a level of indirection between event-based communication and the underlying multicast communication and allows runtime reconfiguration. The IP multicast service is also capable of handling some of the runtime changes. Our approach, of manually specifying a mapping, is valuable when the mapping specification needs relatively few updates in order to maintain efficiency.

The problem of distributing a new mapping specification to all servers is not addressed in this paper. But in order to maintain the semantics of the service, it is important that either all servers change the mapping specification or none. This is a well known problem in the distributed systems field for which many techniques exist.

4.5 Robustness

The robustness of our architecture is due to the fact that there are no central computers (ignoring configurations with dedicated servers and thin clients). Each computer hosting one or more clients, also executes part of the event notification service. We use a soft state approach, relying on refresh and timeout mechanisms, in order to handle crashed processes, crashed computers, link failures, etc. Each server (hosted by some computer) periodically informs the other servers (hosted by other computers) about the notifications of interest to any of its clients, e.g. by sending the aggregated subscriptions on a well known multicast address. Each server expires subscriptions which have not been refreshed for some time. The reason for subscriptions not being refreshed, is of no concern. As an example, if a server does not receive any subscriptions for some time due to e.g. a link failure, then no notifications generated by its client(s) are sent. Whenever the failed link comes up and the server starts receiving relevant subscriptions, the server will again start sending notifications.

5 Prototype

Our prototype is based on the event notification service software developed in the SIENA[2] project. More specifically we have extended the software provided in the siena-java-1.4.2.tar.gz package. This software is written in Java and so are our extensions.

In our current prototype, the server software is linked into the client code and executes within the same address space, as illustrated in Figure 2. A server executes the intra process, the intra host, and the intra LAN event notification service software on behalf of clients hosted by the same process. It should be noted that we consider restructuring this software in order to have a single

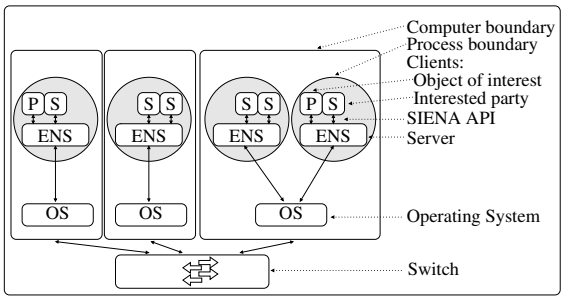

Figure 2: The LAN Event Notification Service

instance responsible for aggregation on a host basis and another part which is executed within each process.

Each server maintains references to its interested parties and their subscriptions. The server also maintains state to keep track of subscriptions made by other servers (on behalf of their clients) in order to determine if a notification is of interest to any other server, hosted by another process on the same computer or another computer on the LAN.

5.1 Intra Process Communication

For intra process communication, a server relies on method calls. Interested parties must implement an interface which define a so-called *notify* method. When a server receives a notification, the server notifies all its clients, for which the subscriptions cover the notification.

5.2 Intra host and Intra LAN Communication

Currently, IP multicast is used to forward notifications from one server to other servers, both within a single computer and between different computers on a LAN.

IP multicast provides some mechanisms which determine if an IP multicast packet is delivered to other processes on the same computer and if the packet is sent out on the LAN. An IP multicast packet is sent out on the LAN if the value of the *time to live* field is 1 or larger. If a so-called *loopback* socket option is set, then a packet is delivered to the other processes on the same computer which have registered interest in this particular IP multicast address. These two mechanisms may be used to forward notifications to other servers hosted by this computer, other servers hosted by other computers on the LAN, or both.

If two servers hosted by the same computer register interest in the same multicast address, then only a single instance of each packet is received by this computer. Aggregation is handled by the multicast software in the operating system, i.e. packets containing event notifications are copied to the different servers by the operating system.

5.3 Subscription Forwarding

Each server aggregates subscriptions on behalf of their interested parties and periodically forwards these subscriptions to all other servers by IP multicast. A separate IP multicast address is used, in order to reduce the risk of operating system buffers being overwritten. Currently, each server forwards its subscriptions independently of the subscriptions made by other servers, i.e. subscriptions are not aggregated, neither on a host basis nor on a LAN basis.

5.4 Packet Senders and Packet Receivers

Servers rely on so-called *packet senders* and *packet receivers* in order to send and receive notifications respectively. An instance of a packet receiver is handled by a separate thread. The thread is waiting for packets on a particular multicast address. A packet sender on the other hand does not have any associated thread, but is executed by the calling thread. Packet senders and packet receivers are handled by a soft state approach, i.e. they are instantiated on demand and timed out whenever not used for some time.

5.5 Outline of Server Algorithm

Each server performs the following actions:

- Periodically: (1) Forwards aggregated subscriptions to the other servers, (2) time out subscriptions which have not been refreshed, and (3) time out unused packet senders and packet receivers
- When one of its interested parties subscribes: Unless the subscription is covered by earlier subscriptions made by its interested parties, immediately forwards the subscription to the other servers
- When a subscription is received from another server: Stores/resets timeout value for the received subscription
- When a notification is received from one of its objects of interest: (1) Unless the notification is not of interest to any other server, forwards the notification to the multicast address associated with a covering advertisement filter and (2) notifies by method call each of its interested parties which have subscriptions covering the notification
- When a notification is received from one of the other servers: Notifies its interested parties which have subscriptions covering the notification, by method calls
- On demand: (1) Instantiates packet senders/receivers or (2) updates mapping specification

6 Empirical Results

In the following we describe the experiments conducted in order to measure the performance and the scalability of our service.

6.1 Environment

For the experiments, standard dual 2GHz AMD Athlon PCs running the Linux 2.4.19 operating system have been used. The PCs were connected by 100Mbps (Mbits per second) switched Ethernet, provided by a Cisco Catalyst 2950XL switch. The switch was configured with IGMP snooping enabled, a technique where the switch maps network layer multicast to link layer multicast by looking for e.g. IGMP host join messages encapsulated within the IP part of packets. The computers were equipped with Intel Ethernet Pro 100 and 3Com 3c905C network interface cards. The software was compiled and executed by a standard Java edition from SUN, version 1.4.1-b21.

6.2 Experiments

For the experiments some client software was written - the object of interest and the interested party code. Each notification had the following attributes: source, type, sequence number, and array. The mapping specification had four advertisement filters, where the source and the type attributes and their values were used to map notifications to potentially four different IP multicast addresses. Interested parties used the sequence number value for measuring the number of

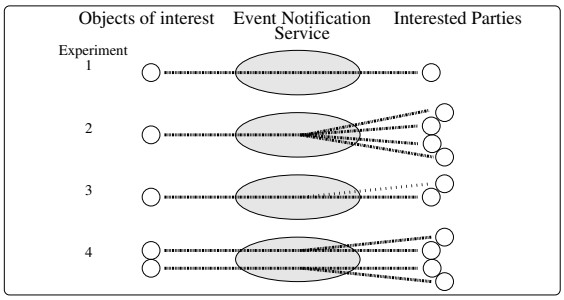

Figure 3: The Different Experiment Configurations

notifications received per second. The length of the array was used to adjust the size of the notifications. A maximum size of 1450 bytes/notification was chosen in order to avoid fragmentation in the protocol stack. The publishing rate for the object of interest was configurable.

The different experiment configurations are illustrated in Figure 3. Each experiment was expected to give information about a certain aspect - (1) the throughput, when notifications are forwarded from an object of interest to a single interested party, (2) the scalability, when several interested parties register interest in the same notifications, (3) the ability to support interested parties with different needs, and (4) the ability to map parts of the "event notification space" to different multicast addresses in order to provide isolation between different (parts of) applications.

Note that the mapping specifications and the subscription filters used in the following experiments have been configured manually to test the performance potential of our service. Clearly, poorly chosen mapping specifications may have reduced the performance.

6.2.1 Experiment 1: One to One Throughput

In the first experiment, a single interested party subscribed to the notifications generated by a single object of interest. The purpose of this experiment was to measure the maximum number of notifications per second transfered between an object of interest and an interested party, for the the intra LAN, the intra host, and the intra process cases. Each client was hosted by a process which also hosted an instance of the server. In the intra process case, a single server was shared between the object of interest and the interested party.

In order to avoid buffer overruns and loss of notifications for the intra LAN and intra host cases, the size of the socket buffers in the operating systems were increased to 2 MBytes. Without this increase lots of notifications were lost, especially for large sized notifications.

The object of interest (and its server) was capable of publishing roughly twice as many notifications per second as the interested party (and its server) was able to receive. Therefore, the publication rate was reduced for the intra host and the intra LAN cases, in order to match the maximum receive rate of the interested party.

The measured throughput, in notifications per second, is given in Table 6. The intra process case provides best performance, although only a single CPU is utilized. The thread which invokes the publish method executes both the server code and the notify method of the interested party. For the intra host and the intra LAN experiments, two CPUs were utilized concurrently, either on the same computer or on different computers.

For the intra LAN and the intra host cases a maximum of approximately 6.5 MBps (MBytes per second), roughly 52Mbps, was measured. These tests were CPU bound, limited by the maximum receive rate of the interested party. Note that more than half the network link capacity was utilized. For comparison, a television quality MPEG-2 encoded video, Main profile in the Main Level, 720 pixels/line $*$ 576 lines, requires maximum 15Mbps[13].

Locality	Notification size in bytes			
	100	500	1000	1450
	Notifications received per second			
Intra LAN	9000	7000	5500	4500
Intra Host	9000	7000	5500	4500
Intra Process	115000	100000	85000	75000

Table 6: The Maximum Number of Notifications Received per Second

6.2.2 Experiment 2: One to Many Scalability

The purpose of the second experiment was to measure the scalability of the service. Four interested parties, each hosted by a separate computer, subscribed to and received the same notifications.

For this experiment, the measured numbers of notifications received per second by each interested party, were *the same* as in the intra LAN case in the first experiment. Hence, the three additional subscribers did not affect the server executing on behalf of the object of interest, neither with respect to processing nor with respect to network bandwidth consumption. The switch was able to handle the copying of packets to the appropriate ports.

It should be noted that if the event notification service had not been able to utilize multicast, the computer hosting the object of interest would have become IO bounded, i.e. the maximum rate of the network link would have been exceeded. In the 1450 bytes/notification case, a unicast-based service would have hit an IO bottleneck even for only two interested parties. The aggregated data rate for the four interested parties in the 1450 bytes/notification case was 26.1 MBps (4 * 4500 notific./sec. * 1450 bytes/notific.).

6.2.3 Experiment 3: One to Many Heterogeneity

The purpose of the third configuration illustrated in Figure 3 was to verify that our event notification service is able to support interested parties hosted by heterogeneous computers and/or network connections. Each client was hosted by a separate computer. One of the interested parties subscribed to and received only some of the event notifications, i.e. only notifications with a particular value for the type attribute. The mapping specification used, mapped these notifications to a separate IP multicast address.

The measurements confirmed that consumption of both network bandwidth and processing resources were reduced accordingly for this interested party and its server.

6.2.4 Experiment 4: Many to Many Isolation

The purpose of the forth experiment illustrated in Figure 3, was to verify that different (parts of) applications may be isolated by using an appropriate mapping specification. Two objects of interest generated notifications with different source attribute value. The mapping specification used, mapped notifications with different value for the source attribute to different IP multicast addresses. The notifications from each object of interest were received by two interested parties. Each client was hosted by a separate computer.

The measured numbers of notifications per second, received by each interested party, were the same as in the intra LAN case in the first experiment. The aggregated publishing rate for the 1450 bytes/notification experiment was 13.05 MBps (2 * 4500 notific./sec. * 1450 bytes/notific.). For a 100Mbps LAN based on broadcast technology, the throughput most likely would have been reduced. This indicates the strength of our event notification service when coupled with switched LAN technology with native multicast support.

7 Further Work

In our further work, we will develop an algorithm for calculating mapping specifications in order to handle dynamic changes in applications and the environment in a more adaptable way. The input to such an algorithm may include the number of multicast addresses, the LAN characteristics (e.g. broadcast, switched), information about imperfections in network to link layer multicast mapping (e.g many to one), some statistics about the past as well as the likely future. The information about the past may be provided by each server measuring and generating statistics about the notifications received and required and the notifications received but discarded. The information about the future may be QoS parameters included in advertisements made by objects of interest, e.g. notification rate, size, and distribution.

We plan to enhance our event notifications service to concurrently utilize a combination of different protocols. The motivation is that different parts of applications have different requirements with respect to e.g. throughput, reliability, and delay. Clients may then indicate QoS parameters in subscriptions and advertisements.

We also would like to avoid the broadcast of subscriptions between servers. A server could hold back subscriptions already covered by subscriptions made by other servers. This is similar to the approach used by IGMP, where only one host sends a membership report for a particular IP multicast address during each time interval.

8 Conclusion

Event-based interaction is inherently many to many communication. Therefore, event-based communication does not map well, performance wise, onto one to one communication primitives. The challenge of utilizing network and link layer multicast support for event notification services is well known, but to our knowledge no implementations for content-based publish/subscribe systems exist.

In this paper we have presented the architecture of a distributed content-based event notification service where notifications are mapped onto multicast communication. The service is targeted at usage within a local area network or an administrative domain. We envisage that the service will be connected to a wide area network event notification service by means of a gateway.

A prototype has been implemented and experiments confirm that our service has a potential for providing both high performance and scalability. Objects of interest may publish several thousand notifications, carrying several MBytes of data, per second. For the experiments performed, the service was unaffected by the number of interested parties, due to the ability of the service to take advantage of native multicast support in network and end system devices.

The scalability and performance allows new application domains to take advantage of event-based interaction. The performance is e.g. more than sufficient for real-time streaming of very high quality video. Application domains requiring parallel processing, e.g. real-time content analysis, may also take advantage of such a service.

9 Acknowledgments

We would like to thank all persons involved in the DMJ (Distributed Media Journaling) project for contributing to the ideas presented in this paper. The DMJ project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431.

References

[1] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In *Proceedings of ICDCS*, pages 262–272. IEEE, 1999.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[3] D. Chambers, G. Lyons, and J. Duggan. Stream Enhancements for the CORBA Event Service. In *Proceedings of the ACM Multimedia (SIGMM) Conference, Ottawa*, pages 61–69, October 2001.

[4] V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne. Scalable Independent Multilevel Distribution in Multimedia Content Analysis. In *Proceedings of the Joint International Workshop on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS 2002), Coimbra, Portugal*, LNCS 2515, pages 37–48. Springer-Verlag, Nov. 2002.

[5] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Real-time Processing of Media Streams: A Case for Event-based Interaction. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pages 555–562. IEEE Computer Society, July 2002.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR)*, 35:114–131, June 2003.

[7] S. McCanne, M. Vetterli, and V. Jacobson. Low-Complexity Video Coding for Receiver-Driven Layered Multicast. *IEEE Journal of Selected Areas in Communications*, 15(6):983–1001, August 1997.

[8] Object Management Group Inc. CORBA services, Event Service Specification, v1.1. *http://www.omg.org/*, 2001.

[9] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In *Proceedings of Middleware 2000*, LNCS 1795, pages 185–207. Springer-Verlag, 2000.

[10] J. Ott, C. Perkins, and D. Kutscher. A message bus for local coordination. *RFC3259*, 2002.

[11] P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pages 611–618. IEEE Computer Society, July 2002.

[12] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content Based Routing with Elvin4. In *Proceedings of AUUG2K, Canberra, Australia*, June 2000.

[13] R. Steinmetz and K. Nahrstedt. *Multimedia: Computing, Communications & Applications*. Prentice Hall, 1995.

[14] TIBCO Software Inc. TIBCO Rendezvous FAQ. http://www.tibco.com/solutions/products/active_enterprise/rv/faq.jsp, 2003.

Paper V

Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis

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

Published: In Proceedings of the 11th ACM International Conference on Multimedia (MM'03), ACM, pages 21-32, Berkeley, California, USA, November 2003.

Evaluation: In total, 255 papers were submitted to ACM MM'03. Three program committee members reviewed each submission. The paper was updated based on review comments and the final version approved by a shepherd. As a result, 43 full papers were accepted for publication.

Author Contribution: This project article combines results from Paper III, Paper IV, as well as some other relevant results contributed by Granmo. Hence, the contributions of Eide follow from the contributions in Paper III and Paper IV. The conceptual integration was done by all authors, while the writing was primarily done by Granmo, Eide, and Eliassen.

[2] Authors are listed alphabetically

Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis

Viktor S. Wold Eide[1,2], Frank Eliassen[2], Ole-Christoffer Granmo[1,2,3], and Olav Lysne[2]*

[1]University of Oslo
P.O. Box 1080 Blindern
N-0314 Oslo, Norway
viktore,olegr@ifi.uio.no

[2]Simula Research Laboratory
P.O. Box 134
N-1325 Lysaker, Norway
viktore,frank,olavly@simula.no

[3]Agder University College
Grooseveien 36
N-4876 Grimstad, Norway
ole.granmo@hia.no

ABSTRACT

Real-time content-based access to live video data requires content analysis applications that are able to process the video data at least as fast as the video data is made available to the application and with an acceptable error rate. Statements as this express quality of service (QoS) requirements to the application. In order to provide some level of control of the QoS provided, the video content analysis application must be scalable and resource aware so that requirements of timeliness and accuracy can be met by allocating additional processing resources.

In this paper we present a general architecture of video content analysis applications including a model for specifying requirements of timeliness and accuracy. The salient features of the architecture include its combination of probabilistic knowledge-based media content analysis with QoS and distributed resource management to handle QoS requirements, and its independent scalability at multiple logical levels of distribution. We also present experimental results with an algorithm for QoS-aware selection of configurations of feature extractor and classification algorithms that can be used to balance requirements of timeliness and accuracy against available processing resources. Experiments with an implementation of a real-time motion vector based object-tracking application, demonstrate the scalability of the architecture.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*Video analysis*

General Terms

Algorithms, design, measurement, performance

Keywords

Real-time video content analysis, parallel processing, task graph scheduling, event-based communication, QoS and resource management

*Authors are listed alphabetically

MM'03, November 2–8, 2003, Berkeley, California, USA.

1. INTRODUCTION

There is evidence that the need for applications that can analyse concurrently and in real-time the content of multiple media streams, such as audio and video, is increasing. For example, in video content analysis applications including feedback control or interaction such as road traffic control systems, automated surveillance systems, and smart rooms, timeliness is an important aspect [1,6,29]. This requires that content analysis applications are able to process the media streams at least as fast as the data is made available to the application.

Real-time content analysis is an active research field where efficient techniques for e.g. multi-object detection and tracking have been found. In such applications, 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.

The above challenges become even more critical when coordinated content analysis of video data from multiple video sources is necessary. Typically, a parallel processing environment is required for real-time performance.

When building a real-time content analysis application, not only must the processing properties of the application be considered, but also the content analysis properties. Such properties we might refer to as Quality of Service (QoS) dimensions and include dimensions such as timeliness and acceptable error rate.

Statements of QoS must generally be expressed by the application user according to some domain specific QoS model that defines the QoS dimensions for the specific application domain. In order to provide some level of control of the QoS provided, the video content analysis application must be scalable so that requirements of timeliness can be met

by allocating additional processing resources. Furthermore, the selection of analysis algorithms must be resource aware, balancing requirements of accuracy against processing time.

In this paper we present a general architecture of video content analysis applications that includes a model for specifying requirements of timeliness and accuracy. The architecture combines probabilistic knowledge-based media content analysis with resource awareness to handle QoS requirements. Based on an application for real-time tracking of a moving object in a video stream, we demonstrate that the architecture is independently scalable at multiple logical levels of distribution by measuring the performance of the application under different distribution configurations. Furthermore, we present experimental results with an algorithm for resource-aware selection of configurations of feature extractor and classification algorithms. In our prototype implementation the algorithm is used to balance requirements of timeliness and accuracy against available processing resources for the same application as above. Although we in this paper focus on video as input, the architecture itself is not limited to video only. It has been designed to handle and exploit input from any combination of different types of sensors in the same application.

Several frameworks for parallel execution of video content analysis tasks have been developed. For instance, in [25] a multi-agent based system for coarse-grained parallelization and distribution of feature extraction is presented. A fine-grained solution for parallel feature extraction is described in [20] where feature extraction is distributed in a hypercube multicomputer network. Other frameworks for parallel and distributed video analysis include [12, 21, 22, 34]. However, these frameworks lack an explicit QoS-model and do not support balancing of accuracy and timeliness against the available processing resources.

The rest of this paper is structured as follows. First, in Section 2 we present a general architecture for (video) content analysis applications. Then we present a QoS model for such applications in Section 3. In Section 4 we discuss architectural requirements that are important for supporting the QoS model. Based on these requirements, the architecture is presented in Section 5. In Section 6 we present empirical results. Lastly, in Section 7 we conclude and provide some pointers to further work.

2. CONTENT ANALYSIS

A general approach for building content analysis applications is to combine low-level quantitative video processing into high-level concept recognition. Typically, such applications are logically organized as a hierarchy of logical modules each encapsulating a video processing task, as illustrated in Figure 1. A task is a well-defined algorithm involved in the video analysis process.

We define task categories according to their functionality in the system. At the lowest level of the hierarchy there are tasks representing video streaming sources. At the level above, the video streams are filtered and transformed by filtering tasks. The transformed video streams are then fed to feature extraction tasks as video segments (e.g. video frame regions). Feature extraction tasks operate on the video segments from the transformed video streams, and in the case of a video frame region, calculate features such as color histograms and motion vectors. Finally, results from feature extraction tasks are reported to classification tasks higher

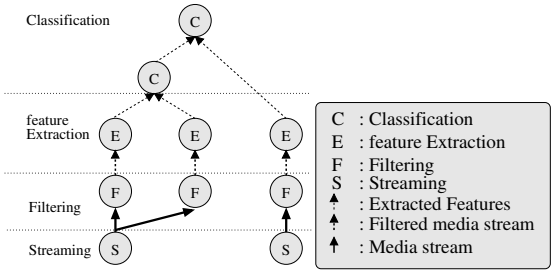

Figure 1: Content analysis hierarchy example.

up in the hierarchy that are responsible for detecting high level domain concepts, such as a moving object in a video stream. In other words, classification is interpretation of extracted features in some application specific context. We will hereafter denote the Streaming, Filtering, feature Extraction, and Classification by the letters S, F, E, and C respectively as seen in Figure 1.

Tasks generally form a directed acyclic graph where the tasks are represented by the nodes in the graph, and the edges represent the directed flows of data between tasks.

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

3. QOS MODEL

In this section we present a QoS model for real-time content based video analysis applications. The model consists of QoS dimensions that we believe characterize the timeliness and accuracy requirements of such applications. Other QoS dimensions such as reliability and availability are considered outside the scope of this work.

The QoS model we have adopted in this work includes the following QoS dimensions: *accuracy*, *temporal resolution*, and *latency*.

The *accuracy* of a media content analysis application can be characterized by its estimated error rate, defined as the number of misclassifications divided by the total number of classifications when analysing a set of media streams. Depending on the media content analysis application, various levels of error rate may be acceptable. For instance, misclassifying events when monitoring an airport for security reasons may be more critical than misclassifying events when indexing a baseball video stream.

The *temporal resolution* dimension specifies the minimum temporal length of an event that the content analysis application should be able to detect. According to the Nyquist sampling theorem [31], any function of time (e.g. stream of high-level concepts) whose highest frequency is W can be completely determined by sampling at twice the frequency, $2W$. In other words, if a stream of high-level concepts is sampled at a frequency less than twice the frequency of the finest temporal details in the stream, high-level concepts may be missed. Hence the value of the temporal resolu-

tion dimension determines the required sampling frequency of the media streams to be analysed.

The *latency* dimension specifies the maximum acceptable elapsed time from an event occur in the real world until it is reported by the appropriate classifier algorithm. For real-time video content analysis applications including feedback control (e.g. road traffic control systems) this dimension is important.

In general, different classes of quality of service can also be identified, varying from best effort service to guaranteed service. The latter class requires support from the system in terms of resource reservation and admission control, while the former does not. Although the problem of resource reservation and admission control have been studied for a long time, their solution has not generally been integrated into more general-purpose operating systems and networks. We therefore restrict the class of processing platforms that we consider to general-purpose ones without special real-time processing features. However, we do assume that we have some level of control over the load of competing applications in the processing environment. Furthermore, we believe our results can easily be adapted to take advantage of processing platforms providing real-time scheduling policies.

4. ARCHITECTURAL REQUIREMENTS

It seems evident that the resource requirements for the application domain of real-time video content analysis are very challenging and will most likely remain so in the near future. This calls for an architecture that is scalable in the sense that the performance of the application scales well with the amount of processing resources allocated to it. Scalability is required in order to be able to cope with increasing QoS requirements and coordinated analysis of an increasing number of media streams.

A scalable application architecture can generally only be obtained by adopting distribution as its basic principle. Scalability of distributed applications is usually achieved by parallelizing application algorithms and distributing the processing of their parts to different processors.

The relative complexity of streaming, filtering/ transformation, feature extraction, and classification depends on the application. Therefore the architecture 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, the feature extraction and the classification may become processing bottlenecks.

A scalable interaction mechanism which also supports such independent parallelization is therefore required.

The level of accuracy that can be supported by a video content analysis application depends on the misclassification behavior (error rate) of the selected configuration of feature extractor and classifier algorithms. Hence configurations of such algorithms must be carefully selected based on the desired level of accuracy.

However, selecting configurations of algorithms which give a high accuracy might result in increased processing time since configurations of algorithms with better accuracy usually require more processing cycles than configurations with poorer accuracy. Therefore, algorithms that can be used to decide whether a QoS requirement can be satisfied in a given distributed physical processing environment are needed. This will include search for an appropriate configuration of feature extractor and classifier algorithms that provides the desired accuracy and that can be allocated to different processors in such a way that the requirements for latency and temporal resolution are fulfilled.

Reduced latency and a smaller temporal resolution may be achieved in a scalable distributed architecture by allocating independent tasks to different processors. A further decrease in temporal resolution may be achieved by deploying *dependent tasks* as pipe-lines also on different processors. E.g., in a maximally distributed video processing pipe-line a frame rate of R may be sustained if no task in the pipe-line has an average processing time per frame exceeding $1/R$.

5. ARCHITECTURE

In this section we develop an architecture that supports the three following content analysis application construction steps:

1. Decomposition of a content analysis application into tasks. The tasks form a task graph as discussed in Section 2. The granularity of the decomposition should be a modifiable parameter because what granularity is appropriate depends on the processing environment at hand, and in particular the number of processors available.

2. Fine grained trading of accuracy against latency (and consequently temporal resolution). The starting point is a "brute force" task graph. By a "brute force" task graph we shall mean a task graph that contains the filtering, feature extraction, and classification tasks deemed relevant, without regard of the processing resources available. Tasks are removed iteratively from the task graph so that either the latency/temporal resolution requirement is met (success) or the accuracy falls below the required level (failure).

3. Scalable deployment and execution of the resulting task graph on multiple processors in a distributed processing environment.

In the next subsections we will first describe some example task graphs. These will be used to exemplify the functionality of the different parts of our architecture. Then, we present the ARCAMIDE algorithm which is used in step two above. Finally, we introduce the architectural basis for applying the ARCAMIDE algorithm. This includes the support of step one and three above. With respect to step three, an event-based interaction mechanism is a key factor for achieving flexible parallelization and distribution.

5.1 Example Task Graphs

Figure 2 illustrates the functional decomposition 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 an algorithm doing video stream decoding and color-to-grey level filtering. The filtered video frame is divided into $m \times n$ blocks (video segments) before a motion estimator calculates motion vectors for the blocks. The block motion vectors are then received by a classification task (a so-called particle filter) and used for object detection and tracking.

We base our example application on motion vector calculation and particle filtering, because these techniques are recent and promising approaches to object/region tracking

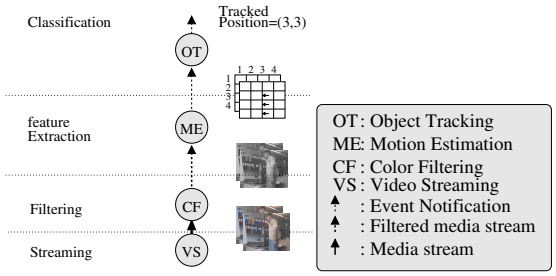

Figure 2: The functional decomposition of the real-time object tracking application.

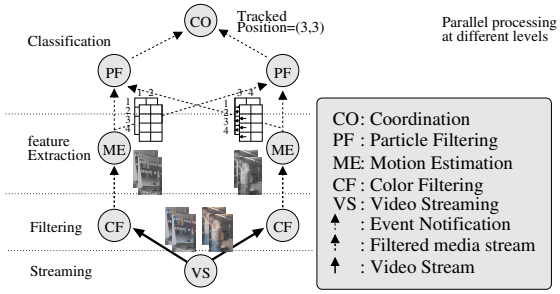

Figure 3: A configuration of the real-time object tracking application where the computation at several levels is parallelized.

in video. To elaborate, calculation of motion vectors (also called optical flow) is a typical pre-processing step in tracking of moving objects/regions [2, 27], and the particle filter is a promising approach to object-tracking which allows e.g. simultaneous tracking and verification [18].

The above content analysis task graph can be executed as a pipeline (each level of the chain is executed in parallel). For instance, the application can be executed on four processors, where the streaming is conducted from one processor, the filtering is executed on a second processor, the motion estimation is conducted on a third processor, and the classification on a forth processor. Such distribution allows an application to take advantage of a number of processors equal to the depth of the hierarchy.

Figure 3 illustrates a task graph with a finer granularity compared to the task graph in Figure 2. The finer granularity has been achieved by independently decomposing the filtering, feature extraction and classification into pairs of two tasks. Such decomposition opens up for focusing the processing resources on the processing bottlenecks at hand. For instance, in Figure 3 the motion estimation could be conducted on two processors while the classification, i.e. particle filtering and coordination (see Section 5.3.5), can be conducted on three processors.

5.2 Determining Accuracy and Latency

In this subsection we describe the ARCAMIDE algorithm [14] for fine grained trading of accuracy against latency (and

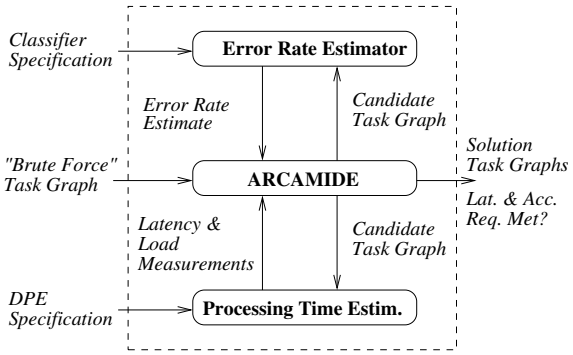

Figure 4: Architecture supporting the accuracy and latency dimension.

hence temporal resolution). Before we go into the details of the ARCAMIDE algorithm, let us provide an overview of the overall ARCAMIDE architecture. As shown in Figure 4, the architecture includes the following functionality:

- An *error rate estimator* which is used to estimate the content analysis (classification) error rate of candidate task graphs, based on a specification of the classification goal (e.g. object tracking). As we will see later, we use so-called dynamic Bayesian networks as a classifier specification language.

- A *processing time estimator* which is used to measure the latency and parallelizability of candidate task graphs, given a specification of the distributed processing environment (DPE) at hand.

- The *ARCAMIDE algorithm* which systematically removes tasks from the task graph in order to reduce the latency of the task graph while at the same time trying to minimize the loss of accuracy.

The output of the architecture is a sequence of solution task graphs ordered so that the estimated latency is decreasing while the estimated error rate is increasing. In this sequence, the first task graph which meets the latency requirement, must also meet the accuracy requirement. Otherwise, the specified DPE will not be able to support the given latency and accuracy requirements. In the following we will first discuss the inputs of the architecture, and then the estimator- and ARCAMIDE functionality in more detail.

5.2.1 The Task Graph and the DPE Specification

We assume that the task graph is annotated with the estimated processing time of each task in milliseconds on the available classes of processors. Furthermore, each edge is assumed to be annotated with the size in bytes of the data communicated between the respective tasks. When this is done for the available streaming, filtering, feature extraction, and classification tasks, we get the "brute force" task graph which the ARCAMIDE algorithm takes as input.

The DPE specification consists of the number and class of processors as well as the network latency and bandwidth between each pair of processors. To simplify the estimation of

the latency of a task graph, we make the following assumptions about the communication between each pair of processors: the communication is contention free and the network latency and bandwidth are constant. These assumptions are introduced to avoid the additional complexity caused by communication contention, routing, etc. (which are not the focus of this paper) while still handling a significant class of DPEs (e.g. dedicated homogeneous computers connected in a dedicated switched LAN).

5.2.2 Dynamic Bayesian Networks and Error Rate Estimation

Dynamic Bayesian networks (DBNs) [16] represent a particularly flexible class of pattern classifiers that allows statistical inference and learning to be combined with domain knowledge. Indeed, DBNs have been successfully applied to a wide range of video content analysis problems [5, 13, 26]. The successful application of DBNs can be explained by their firm foundation in probability theory, combined with the effective techniques for inference and learning that have been developed.

In order to be able to automatically associate high-level concepts (e.g. object position) to video segments, a DBN can be trained on manually annotated video streams. Generally stated, the training is based on finding a more or less accurate mapping between feature space and high-level concept space, within a hypothesis space of possible mappings.

After training the DBN on manually annotated video streams (the training set), the DBN can be evaluated by estimating the number of misclassifications on another manually annotated video stream not used for training (the test set). We shall by the *estimated error rate* of a DBN mean the number of misclassifications divided by the total number of classifications on the test set. This estimated error rate can be seen as a measure on how accurately the DBN will index novel video streams.

One important reason for basing the classification on DBNs is the fact that DBNs can classify even when features are missing. In contrast, other types of classifiers, like neural networks and decision trees, must be retrained whenever the feature set is changed. Accordingly, by using DBNs we make the exploration of the space of possible task graphs and thereby feature subsets more efficient.

5.2.3 The Processing Time Estimator

In this section we describe a simple scheduling algorithm targeting the class of task graphs and DPEs defined in Section 5.2.1. The algorithm is based on generic task graph scheduling principles, as described in [17], adopted to suit the needs of the ARCAMIDE algorithm. That is, we propose measures of latency and parallelizability. These measures are used to guide the ARCAMIDE algorithm.

The goal of the scheduling algorithm is to minimize the estimated latency of the task graph when allocating and scheduling the tasks to the available processors in the DPE. Based on the allocation and scheduling of tasks, the resulting latency and temporal resolution can be estimated. Furthermore, the parallelizability of the task graph can be measured. The respective procedures are summarized in Figure 5 and described below.

We examine the scheduling algorithm first. Let *entry tasks* be tasks without parents and let *exit tasks* be tasks without children in the task graph. We define the *b-level* of a task

```
; The scheduling algorithm
schedule(Tasks)
        ; Initially all tasks are unallocated
        A := ∅;
        N := Tasks;

        WHILE #N > 0 DO
               ; Identifies task with largest b-level
               n_max := ARGMAX n IN N: b_level(n);
               ; Identifies the processor which allows
               ; the earliest start time
               p_min := ARGMIN p IN P: start_time(n_max, p);
               ; Allocates n_max to p_min
               allocate(n_max, p_min);
               A := A ∪ {n_max};
               N := N \ {n_max};
        ; Returns the ready time of each processor
        RETURN {ready_time(p_1), ..., ready_time(p_P)};

; The latency estimation procedure
le(Tasks)
        RETURN MAX(schedule(Tasks));

; The parallelizability measurement procedure
par(Tasks)
        RETURN STD_DEV(schedule(Tasks));
```

Figure 5: The scheduling, latency, and parallelizability procedures.

to be the length of the longest path from the task to an exit node. Likewise, the *s-level* of a task is the length of the longest path from the task to an entry node. The *length of a path* is simply the sum of the task processing times (as given by the task graph) on the path. If multiple classes of processors are available, the average processing time of a task over the available classes of processors is used. Note that when calculating the b-level or the s-level of a task the task itself is included in the path.

A task is either *allocated* to a processor (A) or *not allocated* to a processor (N). Initially, none of the tasks are allocated to processors. At each iteration of the scheduling algorithm, the non-allocated task with the largest b-level is allocated to a processor. This means that execution of long task graph paths are prioritized before execution of short task graph paths. The main reason behind this strategy is that the longest task graph paths often determine the latency of the task graphs when multiple processors are available, and accordingly should be executed as early as possible.

When a task is to be allocated to a processor the task is scheduled at the earliest *start time* possible. The task may be started when the processor becomes available after previous processing, and the task receives the data produced by its task graph parents. The scheduled *stop time* of a task is simply the sum of its scheduled start time and its processing time (specified by the task graph). A task receives data from a task graph parent at the scheduled stop time of the parent if the two tasks are located on the same processor. Otherwise, the communication time of the data must be added to the data receive time.

When allocating the non-allocated task with the largest b-level to a processor, the processor which allows the earliest task start time is selected. This corresponds to a greedy

step towards the goal of minimizing the estimated latency of the task graph. Consequently, the processor selection is determined by the location of the tasks parents in the task graph as well as the communication time of the corresponding data.

This iteration continues until all the tasks have been allocated. Finally, the scheduled stop time, $ready\_time(p_i)$, of the last task to be executed on each processor is returned.

To conclude, the estimated latency of the task graph corresponds to the largest processor ready time. Also, by taking the standard deviation of the processor ready times, we measure how well the scheduling algorithm was able to balance the processing load on the available processors. These two measurements are used by the ARCAMIDE algorithm to search for task graphs with low latency, and which take advantage of the available processors without considering pipelining.

5.2.4 The ARCAMIDE Algorithm

In this section we describe a heuristic search algorithm, the ARCAMIDE algorithm, which prunes a "brute force" task graph in order to offer tradeoffs between estimated error rate and latency. An important goal in this context is to take advantage of the available processors so that the temporal resolution is maximized.

Note that the ARCAMIDE algorithm does not remove the classification tasks from the task graph. Without classifiers the task graph will only output low-level features and no high-level concepts. This means that the classifier tasks should be treated as a special case. Therefore, we partition the tasks (in the brute force task graph) into Streaming, Filtering and feature Extraction tasks, hereafter denoted SFE-tasks, and classification tasks, hereafter denoted C-tasks.

If the search for a task graph that does not violate the QoS requirements is to be computationally practical, only a very small number of the possible task graphs may be evaluated. E.g. if there are no edges in a task graph containing n SFE-tasks, there are 2^n possible candidate subgraphs. Consequently, the choice of which subgraphs to evaluate is of paramount importance.

In contrast to evaluating all the possible subgraphs of the "brute force" task graph, the ARCAMIDE algorithm consists of two task selection stages. In both stages of the algorithm, the most inefficient parts of the task graph (when considering estimated error rate and latency) are pruned. Roughly stated, this procedure corresponds to a standard sequential backward feature subset search (see [7]), extended to handle task graphs. The task graph search is performed backwards, rather than forwards, in order to avoid a computationally expensive n-step look-ahead search, made necessary by the task graph data dependencies, and in some cases by complexly interacting features. Obviously, other feature subset search procedures can be applied by the ARCAMIDE algorithm (such as beam search [24], genetic algorithms, or branch and bound search [7]) by extending them to handle task graphs. However, due to its simplicity, computational efficiency and goal-directed behavior this paper applies a sequential backward task graph search procedure.

The ARCAMIDE algorithm (see Figure 6) takes as input a set of SFE-tasks, a set of C-tasks, a task irrelevance threshold i, and a parallelizability threshold c. The irrelevance threshold is used for removing tasks irrelevant to the content analysis goal. That is, the error rate of each SFE-task

```
; The ARCAMIDE algorithm
arcamide(SFE, C, i, c)
   ; Stage 1
   FOR EACH t IN SFE
      ; Removes t if irrelevant
      IF er({t} ∪ desc(t, SFE)) > i THEN
                SFE := SFE \ ({t} ∪ desc(t, SFE));
   ; Stage 2
   WHILE #SFE > 0 DO
      ; Determines whether critical paths
      ; need to be shortened
      IF par(SFE ∪ C) > c THEN
                ; Shortens critical paths
                SFE' := ARGMAX t IN SFE:s_level(t);
                t_max := ARGMAX t IN SFE':ef(t, SFE);
      ELSE
                t_max := ARGMAX t IN SFE:ef(t, SFE);

      ; Removes t_max and its
      ; task graph descendants
      SFE := SFE \ ({t_max} ∪ desc(t_max, SFE));

      ; Outputs SFE ∪ C, error rate, and
      ; processing time of F
      OUTPUT <SFE ∪ C, er(SFE), le(SFE ∪ C)>;

   RETURN;
```

Figure 6: The ARCAMIDE algorithm.

(and its task graph descendants) is estimated individually. If the resulting estimated error rate is not significantly better than what is achieved with pure guessing, i, the SFE-task is removed along with its descendants. This completes stage one of the ARCAMIDE algorithm.

The parallelization threshold c controls whether critical paths are to be shortened or the least efficient tasks are to be removed in stage two of the ARCAMIDE algorithm. We take $par(SFE \cup C) > c$ as an indication that the scheduling algorithm is not able to take advantage of the available processors due to the length of critical paths. Then, the only way to reduce the latency is to remove SFE-tasks at the end of these paths:

$$SFE' := \underset{t \in SFE}{argmax}\ s\_level(t).$$

If, on the other hand, $par(SFE \cup C) \leq c$ we take this as an indication that the scheduling algorithm is able to balance the processing load between the available processors. Accordingly, the least "efficient" SFE-task should be removed from the task graph along with its descendants.

Among the tasks considered for removal, the task t that maximizes the efficiency function

$$ef(t, SFE) \equiv \frac{er(SFE) - er(SFE \setminus (\{t\} \cup desc(t, SFE)))}{le(\{t\} \cup desc(t, SFE))}$$

is removed from SFE in addition to its task graph descendants in SFE, $desc(t, SFE)$. Here, $er(T)$ denotes the estimated error rate of task set T, and $le(T)$ denotes the estimated latency of task set T. In short, the efficiency function rewards tasks which contribute to maintaining the estimated error rate of SFE and which in addition are computationally cheap.

When the ARCAMIDE algorithm stops, it has generated a sequence of task graphs with different estimated error rate/latency tradeoffs. These task graphs are sought configured to fully utilize the specified processing environment without considering pipelining. The task graph which best fits the provided QoS requirements (in terms of latency, temporal resolution, and accuracy) is selected for deployment in the actual processing environment, as discussed in the following subsection.

5.3 Scalability

The distributed and parallel processing of one of the task graphs suggested by the ARCAMIDE algorithm enables that QoS requirements can be satisfied by throwing processing resources at the problem. In our architecture components are the unit of deployment. The scalability of the overall application is determined by the scalability of the inter component communication mechanism and the scalability provided at the different levels, i.e. video streaming, filtering/transformation, feature extraction, and classification [8]. We now describe how scalability is achieved in each of these cases.

5.3.1 Event-based Component Interaction

An event-based interaction mechanism enables scalable and independent parallelization of the different levels in the content analysis hierarchy, as described in the following. From Figure 1, 2, and 3, it should be clear that components interact in different ways, such as one to one, one to many (sharing or partitioning of data), many to one (aggregation), and many to many.

In [9], we argue that the requirements for the real-time content analysis application domain fit very well with the publish/subscribe interaction paradigm, leading to an event-based interaction model. Event-based interaction provides a number of distinguishing characteristics, such as asynchronous many to many communication, lack of explicit addressing, indirect communication, and hence loose coupling. Event-based systems rely on some kind of event notification service. A distributed event notification service is realized by a number of cooperating servers. Clients connect to these servers and are either *objects of interest*, *interested parties*, or both. An object of interest publishes event notifications, or just notifications for short. In *content-based* publish/subscribe systems, such as [3, 28, 30, 33], a notification may be a set of type, name, and value tuples. Interested parties subscribe in order to express interest in particular notifications. A subscription is an expression with constraints on the names and values of notifications. The responsibility of the event notification service is routing and forwarding of notifications from objects of interest to interested parties, based on content, i.e. the type, name, and value tuples. The servers jointly form an overlayed network of content-based routers. A survey of the publish/subscribe communication paradigm and the relations to other interaction paradigms are described in e.g. [11].

In our architecture, the different components connect to and communicate through the event notification service, as illustrated in Figure 7 for the application configuration in Figure 3. As a consequence, a component does not need to know if notifications have been generated by a single or a number of components or the location or the identity of the other components. The bindings between components are loose and based on what is produced rather than by whom.

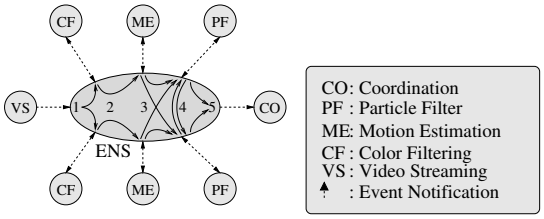

Figure 7: Inter component communication for the configuration in Figure 3. Components interact through an Event Notification Service, labeled ENS.

The what- rather than whom-characteristics of event-based communication is a key factor for achieving the flexible parallelization and distribution. As an example, assume that each PF component subscribes to notifications covered by the following subscription:

```
src=vs func=me
```

Assume further that each ME component publishes the calculated motion vectors as notifications:

```
src=vs func=me time=[t,dt] block=[1,1] vector=[ 0 ,0]...
src=vs func=me time=[t,dt] block=[3,2] vector=[-1 ,0]...
src=vs func=me time=[t,dt] block=[4,4] vector=[ 0 ,0]
```

The event notification service is then responsible for forwarding these notifications to the PF components, as indicated by label 3 in Figure 7. As a result, from a ME component's point of view it does not matter if there is a single or a number of components interested in the published notifications. Similarly, from a PF component's point of view it does not matter if the motion vectors have been calculated by a single or a number of ME components. This illustrates that event-based interaction enables independent parallelization of the different levels in the content analysis hierarchy. In Section 6, we present some performance numbers for a scalable distributed content-based event notification service, which is able to take advantage of native IP multicast support.

5.3.2 Video Streaming

Real-time video is quite challenging with respect to processing requirements, the massive amounts of data, and the imposed real-time requirements. A video streaming source which must handle each and every interested component individually will not scale. Therefore, the sender side processing and network bandwidth consumption should be relatively unaffected by the number of receiving components.

Scalable one to many communication is what IP multicast has been designed for. However, sending a full multicast video stream to all receivers wastes both network and receiver processing resources when each receiver only processes some regions in each video frame. In [23], 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

and/or temporarily. Parallel processing poses a related kind of heterogeneity challenge, but an additional motivation is the distribution of workload by partitioning data. When using an event notification service for video streaming, as described in [4, 9, 32], the video streaming component may send different blocks of each video frame as different notifications. By publishing each video frame as a number of notifications, interested parties may subscribe to only a certain part of a video stream and thereby reduce resolution both spatially and temporally. This strategy, coupled with an event notification service capable of mapping notifications onto IP multicast communication, provides scalable video streaming.

5.3.3 Filtering and Transformation

If the representation, the spatial resolutions, or the temporal resolutions offered by a S component is not appropriate for an E component, filtering and transformation is necessary. Filtering and transformation bridge the gap between what a S component offers and an E component can handle. Scalable filtering and transformation requires that an F component may process only some part of a video stream. Therefore, in our approach an F component may subscribe to only some blocks of the video stream, both spatially and temporally, illustrated in Figure 7, labeled 1. The filtered and transformed blocks are then published as notifications, labeled 2 in the same figure. As a result, filtering and transformation is efficiently distributed and parallelized. This is also illustrated in Figure 3, where the left and the right part of each video frame is received by different motion estimation components.

5.3.4 Feature Extraction

A feature extraction algorithm operates on video segments from the filtering and transformation level (e.g. video frame blocks) and extracts quantitative information, such as motion vectors and color histograms.

A scalable solution for feature extraction requires that E components may process only some part of a filtered video stream. 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. Additionally, the algorithms may be arbitrarily complex, in general.

Feature extraction algorithms for video, such as calculations of motion vectors, color histograms, and texture roughness, often operate locally on image regions. In our architecture spatial parallelization and distribution of such feature extractors are supported by a block-based approach.

Our implementation of a motion estimation component allows calculation of motion vectors for only some of the blocks in a video frame. In Figure 8, the motion vectors calculated by a single component have been drawn into the video frame. 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 calculated motion vectors are published as a notifications. The event notification service forwards each notification to the interested subscribers, as illustrated by label 3 in Figure 7.

Figure 8: Block-based motion estimation example.

5.3.5 Classification

The final logical level of our architecture is the classification level. At the classification level each video 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.

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 processor, but accurately tracking the position of multiple people as well as their interactions (talking, shaking hands, etc.) could require several processors. Multiple video streams may increase the content analysis complexity even further. In short, when the classifier is running on a single processor, the classification may become the processing bottleneck of the content analysis application.

The particle filter (PF) [19] is an approximate inference technique that allows real-time DBN-based video content analysis. In the following we briefly describe our use of the PF in more detail. Then we propose a distributed version of the PF, and argue that the communication and processing properties of the distributed PF allow scalable distributed classification, independent of distribution at the other logical levels.

Our PF is generated from a dynamic Bayesian network specifying the content analysis task. During execution the PF partitions the video stream to be analysed into time slices, where for instance a time slice may correspond to a video frame. The PF maintains a set of particles. A single particle is simply an assignment of a content class to each video 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

Figure 9: The center position of the tracked object, calculated by the coordinator, has been drawn as a white rectangle.

means that multiple feature interpretations can be maintained concurrently in time, ideally until uncertainty can be resolved and noise can be suppressed. When a new time slice is to be analysed, each particle is independently extended to cover new video segments, driven by the dynamic Bayesian network specification. In order to maintain a relevant set of particles, unlikely particles are then 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 PF procedure.

In order to support scalability, we propose a distributed version of the PF. The particles of the single PF are parted into n groups which are processed on n processors. An event based communication scheme maintains global classification coherence. The communication scheme is illustrated in Figure 7 and discussed below. n 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 PF 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 Figure 7). Then, each PF component executes the PF 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 video segment content classes to the coordinator (label 5 in Figure 7) which estimates the most probable content class of each video segment. E.g., in Figure 9 the output of the CO component has been drawn based on the object position likelihoods produced by the PF components.

In the above communication scheme only $2n+1$ messages are submitted per time slice, relying on native multicast support in the event notification service. As shown empirically in [15], by only submitting a single particle per message no loss of accuracy is detected in the object tracking case.

6. EMPIRICAL RESULTS

In this section, we first present some empirical results for the ARCAMIDE algorithm, when applied to our object tracking application. Then we present the measured performance for this application, when executed by a varying number of processors. Lastly, performance numbers are presented for our distributed content-based event notification service, which should reduce an observed bottleneck problem for the first implementation of this application.

6.1 Accuracy and Latency

We now modify the object tracking example application from Figure 3 to evaluate the ARCAMIDE algorithm empirically. First of all, we assume that each video frame in the video stream is partitioned into 8×8 video frame blocks. The modification consists of adding a color histogram calculation task and a texture calculation task to each video frame block. We assume that the motion estimation in a video frame block depends on color-to-grey level filtering of that block. Finally, the goal of the content analysis application is refined to recognition of whether an object passes from the left to the right or from the right to the left.

Thus, we have five different types of SFE-tasks related to each video frame block: streaming, color-to-grey level filtering, motion estimation, texture calculation, and color histogram calculation. These have different content analysis and processing characteristics. In our simulation, when using motion to detect an object in a video frame block, the probability of false positives and false negatives are assumed to be 0.3. Likewise, when using texture the probability of false positives is assumed to be 0.5 and the probability of false negatives is assumed to be 0.3. When using color the latter probabilities are reversed. Color-to-grey-level filtering only produces intermediate results used in the motion estimation. Obviously, the specified probabilities depend on the environment (e.g. sunshine, darkness) and are here set to reflect rather difficult environment conditions. Finally, the processing time of the streaming task is set to 3 ms, and the processing time of the other tasks are set to 1 ms. The transmission time of a video frame block (either color-to-grey level filtered or not) across the network is considered to be 1 ms.

To evaluate the ARCAMIDE algorithm we trained two 10-state Hidden Markov Models (one for each object passing direction) on 1000 simulated video frame sequences of objects moving from the left to the right and 1000 simulated video frame sequences of objects moving from the right to the left. Here, an object appears on average twice in each video frame block of the two center rows when passing the camera view. We used another independent set of simulated video frame sequences (of identical size) to prune the task graph.

When trading off estimated error rate and latency, the ARCAMIDE algorithm behaves as shown in Figure 10, 11, and 12 respectively for 1 processor, 10 processor, and 100 processor environments. When selecting SFE-tasks for 1 processor, we see from Figure 10 that the estimated latency initially can be reduced with little increase in estimated error rate (while inaccurate SFE-tasks are removed). However, when e.g. SFE-tasks near the first and eighth video

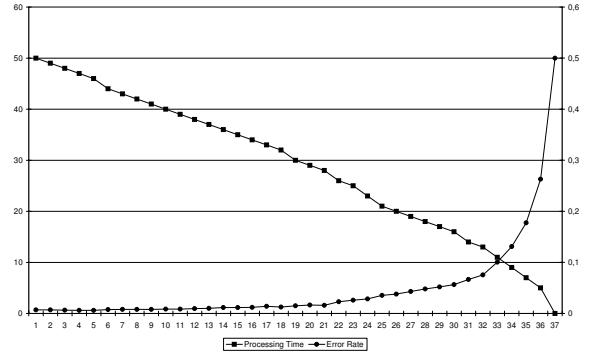

Figure 10: The estimated latency and error rate (y-axis) after each task removal (x-axis) - 1 CPU.

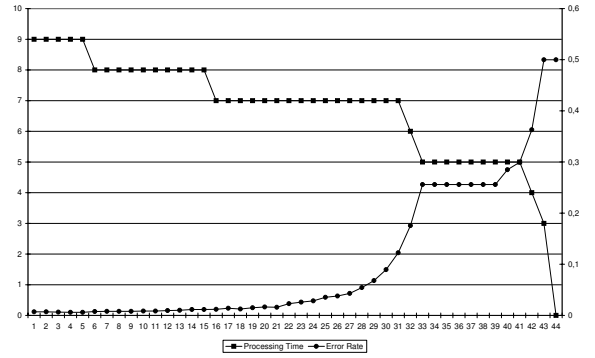

Figure 11: The estimated latency and error rate (y-axis) after each task removal (x-axis) - 10 CPUs.

frame block columns (the object entry regions) must be removed, the estimated content analysis error rate increases more dramatically. When introducing 10 processors, we see in Figure 11 that the estimated latency is reduced in steps — the processing time of all the processors must be reduced before the latency can be reduced. Also note that the data dependencies between color-to-grey level filtering and motion estimation make these tasks the target of removal from removal number 28 and thereafter. When considering 100 processors (Figure 12), initially only the removal of motion SFE-tasks will reduce the processing time due to the dependency of motion SFE-tasks on color-to-grey level filtering tasks. As seen in Figure 12, there are mainly two interesting task graphs; one containing all the relevant SFE-tasks and one containing only texture and color SFE-tasks.

6.2 Scalability of Object Tracking Application

In order to examine the scalability of our architecture, five object tracking configurations were used — targeting 1, 2, 4, 8, and 10 processors respectively. A separate computer hosted the video streaming component. The configurable parameters of the motion estimation component (e.g. the search area) and the particle filtering component (e.g. the

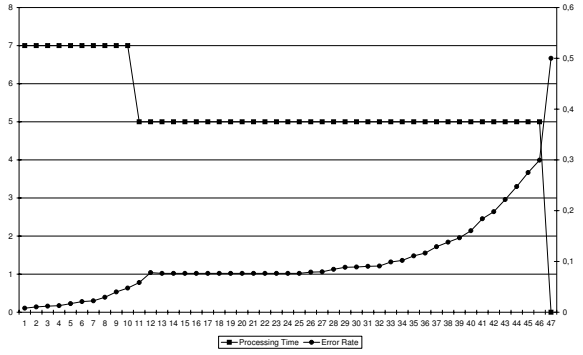

Figure 12: The estimated latency and error rate (y-axis) after each task removal (x-axis) - 100 CPUs.

Table 1: The Measured Number of Frames per Second for Different Configurations of the Real-time Object Tracking Application

	Number of processors				
	1	2	4	8	10
	Frames per second				
Ideal Frame Rate	2.5	5	10	20	25
Streaming	2.5	5	10	20	25
Filtering/Feature Extraction	2.5	5	8.5	13.5	16
Classification	2.5	5	10	20	25

number of particles) were set so that they had similar processing resource requirements. The process hosting a motion estimation component also always hosted a video decoding and filtering component.

The first configuration was similar to the configuration in Figure 2. One video decoding and filtering component, one motion estimation component, one particle filter component, and one coordination component were all executed on a single processor. In the second configuration this pipeline was executed on two processors, that is, the filtering and motion estimation components were executed on one processor and the particle filter and coordination component were executed on another processor. In order to take advantage of additional processors, new configurations were created by stepwise adding one motion estimation component (and implicitly also one filtering component) and one particle filter component, each executed by a dedicated processor. The configuration illustrated in Figure 3 was executed on 4 processors. In the 10 processor configuration, five motion estimation components and five particle filtering components were used.

For the experiments, standard 1667MHz dual AMD Athlon PCs running the Linux operating system have been used. The PCs were connected by 100Mbps switched Ethernet. Additionally, standard IP multicast video streaming was used in this implementation.

The achieved frame rate, i.e. the temporal resolution, for each configuration is shown in Table 1. The frame rate increased linearly with the number of processors, except for the filtering and motion estimation part of the computation.

Table 2: Maximum Number of Notifications Received per Second for Notifications of Different Sizes

Locality	Notification size in bytes			
	100	500	1000	1450
	Notifications received per second			
Intra LAN	9000	7000	5500	4500
Intra Host	9000	7000	5500	4500
Intra Process	115000	100000	85000	75000

This was caused by the IP multicast based video streaming used in these experiments. Each filtering component had to decode and filter the complete IP multicast video stream, despite the fact that the motion estimation component processed only some blocks of each video frame. The ability of the distributed classifier to handle the full frame rate was tested on artificially generated features.

6.3 Event Notification Service

As a solution to the bottleneck observed in the previous experiment, an event-based approach to video streaming has been developed. E.g. a filtering component may register interest in only certain blocks and reduce resolution both spatially and temporally. However, a scalable and high performance event notification service is required in order to provide the necessary throughput. In [10], we present the architecture, the implementation, and the measured performance for our distributed local area network content-based event notification service. The service provides both intra process, intra host, and intra LAN communication.

The throughput for components located on different computers (intra LAN) varied from 4500 notifications per second (1450 bytes each) to 9000 notifications per second (100 bytes each), as illustrated in Table 2. A maximum of approximately 6.5 MBps (MBytes per second) was measured. The performance of the service was unaffected when there were a number of interested parties for the same notifications, due to the fact that the service utilizes native IP multicast support. Additionally, the service is able to isolate different parts of the "event notification space" by mapping notifications to different multicast addresses. As a result, the service is able to handle a number of video streams concurrently.

7. CONCLUSION AND FURTHER WORK

In this paper we have presented a general architecture for distributed real-time video content analysis applications. Furthermore, we have proposed a model for specifying requirements of timeliness and accuracy that can be used to deduce the application's resource requirements from a given QoS specification.

A salient feature of the architecture is its combination of probabilistic knowledge-based media content analysis with QoS and distributed resource management to handle QoS requirements. A further feature is its independent scalability at multiple logical levels of distribution to be able to meet increasing QoS requirements in different QoS dimensions.

This has been achieved by first developing a parallel version of an approximate inference technique known as the particle filter. We demonstrated that the parallel particle filter allows for real-time video content analysis based on dynamic Bayesian networks. Next, we presented an algorithm for balancing the requirements of timeliness and accuracy against available distributed processing resources. We demonstrated its behavior for a real-time motion vector based object tracking application.

Independent scalability at multiple logical levels was primarily achieved through the development of a high performance event notification service as the preferred communication mechanism of the architecture. The scalability was demonstrated through experiments with an implementation of the application mentioned above.

Our future work will include the development of a complete QoS management architecture for real-time video content analysis applications. The work presented here represents steps towards that goal.

8. ACKNOWLEDGMENTS

We would like to thank all persons involved in the DMJ (Distributed Media Journaling) project for contributing to the ideas presented in this paper. The DMJ project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431.

9. REFERENCES

[1] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik. A Real-time Computer Vision System for Measuring Traffic Parameters. In *Computer Vision and Pattern Recognition (CVPR'97), San Juan, Puerto Rico*, pages 495–501. IEEE, June 1997.

[2] A. Bors and I. Pitas. Prediction and tracking of moving objects in image sequences. *IEEE Transactions on Image Processing*, 9:1441–1445, August 2000.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[4] D. Chambers, G. Lyons, and J. Duggan. Stream Enhancements for the CORBA Event Service. In *Proceedings of the ACM Multimedia (SIGMM) Conference, Ottawa*, October 2001.

[5] S.-F. Chang and H. Sundaram. Structural and Semantic Analysis of Video. In *Multimedia and Expo 2000 IEEE*, volume 2, pages 687–690, 2000.

[6] C. Chen, Z. Jia, and P. Varaiya. Causes and Cures of Highway Congestion. *Control Systems Magazine, IEEE*, 21(6):26–32, December 2001.

[7] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), 1997.

[8] V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne. Scalable Independent Multi-level Distribution in Multimedia Content Analysis. In *Proceedings of the Joint International Workshop on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS), Coimbra, Portugal*, LNCS 2515, pages 37–48. Springer-Verlag, November 2002.

[9] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Real-time Processing of Media Streams: A Case for Event-based Interaction. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pages 555–562. IEEE Computer Society, July 2002.

[10] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Extending Content-based Publish/Subscribe Systems with Multicast Support. Technical Report 2003-03, Simula Research Laboratory, July 2003.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR)*, 35:114–131, June 2003.

[12] A. R. François and G. G. Medioni. A Modular Software Architecture for Real-Time Video Processing. In *Proceedings of the Second International Workshop on Computer Vision Systems (ICVS), Vancouver, Canada*, volume 2095 of *Lecture Notes in Computer Science*, pages 35–49. Springer, July 2001.

[13] A. Garg, V. Pavlovic, J. Rehg, and T. Huang. Integrated Audio/Visual Speaker Detection using Dynamic Bayesian Networks. In *Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000*, pages 384–390, 2000.

[14] O.-C. Granmo. 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)*, pages 124–139. ICEIS Press, April 2002.

[15] O.-C. Granmo, F. Eliassen, O. Lysne, and V. S. W. Eide. Techniques for Parallel Execution of the Particle Filter. In *Proceedings of the 13th Scandinavian Conference on Image Analysis (SCIA2003)*, volume 2749 of *Lecture Notes in Computer Science*, pages 983–990. Springer, June 2003.

[16] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Series for Statistics for Engineering and Information Science. Springer Verlag, 2001.

[17] Y.-K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[18] B. Li and R. Chellappa. A generic approach to simultaneous tracking and verification in video. *IEEE Transactions on Image Processing*, 11:530–544, May 2002.

[19] J. S. Liu and R. Chen. Sequential Monte Carlo methods for Dynamic Systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.

[20] V. Manian and R. Vasquez. A Computational Framework for Analyzing Textured Image Classification. In *IEEE Conference on Intelligent Systems for the 21st Century*, volume 1, pages 717–723. IEEE, 1995.

[21] L. Marcenaro, F. Oberti, G. L. Foresti, and C. S. Regazzoni. Distributed Architectures and Logical-Task Decomposition in Multimedia Surveillance Systems. *Proceedings of the IEEE*, 89(10):1419–1440, October 2001.

[22] J. Martínez, E. Costa, P. Herreros, X. Sánches, and R. Baldrich. A modular and scalable architecture for PC-based real-time vision systems. *Real-Time Imaging*, 9:99–112, April 2003.

[23] S. McCanne, M. Vetterli, and V. Jacobson. Low-complexity video coding for receiver-driven layered multicast. *IEEE Journal of Selected Areas in Communications*, 15(6):983–1001, August 1997.

[24] T. M. Mitchell. *Machine Learning*. Computer Science Series. McGraw-Hill International Editions, 1997.

[25] Y. Nakamura and M. Nagao. Parallel Feature Extraction System with Multi Agents -PAFE-. In *11th IAPR International Conference on Pattern Recognition*, volume 2, pages 371–375. IEEE, 1992.

[26] M. Naphade and T. Huang. Extracting semantics from audio-visual content: the final frontier in multimedia retrieval. *IEEE Transactions on Neural Networks*, 13(4):793–810, 2002.

[27] R. Okada, Y. Shirai, and J. Miura. Object tracking based on optical flow and depth. In *Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 565–571. IEEE, December 1996.

[28] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In *Proceedings of Middleware*, pages 185–207, 2000.

[29] B. Ozer and W. Wolf. Video Analysis for Smart Rooms. In *Internet Multimedia Networks and Management Systems, ITCOM, Denver Colorado USA*, volume 4519, pages 84–90. SPIE, July 2001.

[30] P. R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*. IEEE Computer Society, July 2002.

[31] W. K. Pratt. *Digital Image Processing*. Wiley-Interscience. John Wiley & Sons, Inc., 1991.

[32] T. Qian and R. Campbell. 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, May 1997.

[33] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with Elvin4. In *Proceedings of AUUG2K, Canberra, Australia*, June 2000.

[34] W. Zhou, A. Vellaikal, and S. Dao. Cooperative Content Analysis Agents for Online Multimedia Indexing and Filtering. In *Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications*, pages 118–122, 2001.

Paper VI

Exploiting Content-Based Networking for Video Streaming

Viktor S. Wold Eide, Frank Eliassen, and Jørgen Andreas Michaelsen

Published: In Proceedings of the 12th ACM International Conference on Multimedia, Technical Demonstration, (MM'04), ACM, New York, New York, USA, pages 164-165, October 2004.

Evaluation: No written review was received for this technical demonstration paper. The paper was accepted for publication and a technical demonstration was given at the ACM MM'04 conference by Michaelsen and Eide.

Author Contribution: An initial version of the software described in this paper was written by Eide, while the ideas were discussed with Eliassen. However, this implementation was quite rudimentary and did, e.g., not include any code for compression. Michaelsen and Eide collaboratively developed the software into the state described in this paper, both with respect to design and implementation[3]. Eide was the driving force behind writing this paper, while both coauthors contributed by commenting on draft versions of the paper.

[3] A history file included in the software, tries to document changes and contributions

Exploiting Content-Based Networking for Video Streaming

Viktor S. Wold Eide[1,2] Frank Eliassen[1] Jørgen Andreas Michaelsen[2]

[1]Simula Research Laboratory
P.O. Box 134
N-1325 Lysaker, Norway
{viktore,frank}@simula.no

[2]University of Oslo
P.O. Box 1080 Blindern
N-0314 Oslo, Norway
{viktore,jorgenam}@ifi.uio.no

ABSTRACT

This technical demonstration shows that content-based networking is a promising technology for multireceiver video streaming. Each video receiver is provided with fine grained selectivity along different video dimensions, such as region of interest, quality, colors, and temporal resolution. Efficient delivery is maintained, in terms of network utilization and processing requirements. A prototype demonstrates the feasibility of this approach and is available as open source.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Content-based networking*; H.4.3 [**Information Systems Applications**]: Communications Applications—*Video streaming*

General Terms

Design, experimentation

Keywords

Content-based networking, distributed content-based publish subscribe systems, fine granularity video streaming, scalable video coding, layered video

1. INTRODUCTION

A lot of effort has been made in order to efficiently deliver video data over best-effort networks, such as the Internet [6]. Unicast delivery may provide each client with a customized stream, but is inefficient and does not scale. Multicast delivery, provided at the network level or by overlay networks, may improve network efficiency. However, a single multicast stream provides clients with no selectivity. Simulcast delivery may provide clients with a choice between a few number of streams, each having a different tradeoff between quality characteristics and resource requirements. A combination of layered coding and multicast is also rather course grained, but improves network efficiency as the amount of redundant information in different streams is reduced [7].

However, fine grained selectivity along different video quality dimensions is important for domains such as real-time distributed video content analysis, where timeliness and accuracy requirements may necessitate parallel processing of video data. The video streaming software presented here is part of our framework targeting this application domain [3]. The software exploits content-based networking in order to provide fine grained selectivity and efficient delivery.

MM'04, October 10-16, 2004, New York, New York, USA.
ACM 1-58113-893-8/04/0010.

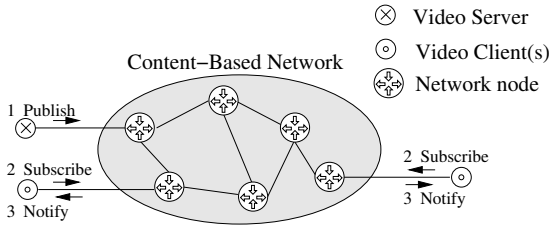

Figure 1: Content-based network example.

2. CONTENT-BASED NETWORKING

In content-based networking [2], messages are forwarded based on content, and not on an explicit address. Each message contains a set of attribute/value pairs and clients express interest in certain messages by specifying predicates over these attribute/value pairs. These predicates are used by the network nodes for routing. Messages are forwarded based on these routing tables and delivered only to clients having matching selection predicates. Filtering is pushed towards the source and each message should traverse a link at most once. Content-based networks are currently realized as overlay networks. Distributed content-based publish subscribe systems [1] are examples of content-based networks. The messages, called notifications, are injected into the network by publishing as illustrated in Figure 1. Clients express their interests in subscriptions as predicates over the attributes/value pairs and are notified accordingly.

3. VIDEO STREAMING

Video servers publish notifications, which encapsulate encoded video data. Each notification contains a subset of the video data for a single frame and is delivered to all video clients having matching subscriptions. The presented video coding software exploits well known video compression techniques, but is implemented from scratch as extending e.g. other MPEG implementations for content-based networking seemed non-trivial. The software is Open Source and is available from http://www.ifi.uio.no/~dmj/.

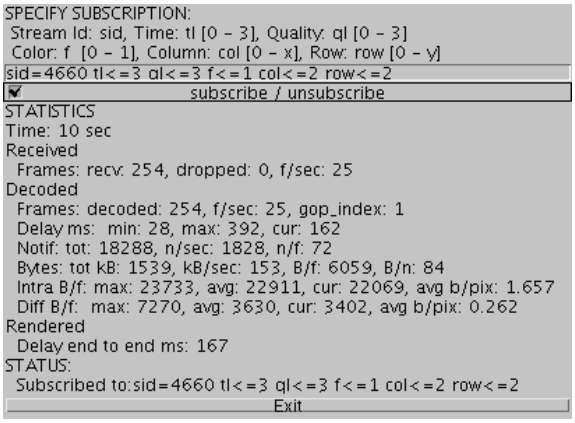

Figure 2: Experimental video client controller.

Figure 3: Four video clients with different interests.

3.1 Fine Granularity Selectivity

Our video coding scheme supports selectivity along the following dimensions: *regional, quality, color, and temporal.*

The smallest selectable *region* is a so-called superblock, which is a number of 16×16 pixel macroblocks.

In the *quality* dimension, a layered coding is used. Each block within a macroblock is transformed to the frequency domain by using the Discrete Cosine Transform. The DCT coefficients are quantized and bit-plane coded [5]. The different bit-planes are mapped to different quality layers. Notifications for the base layer contain the most significant bits.

With respect to *colors*, the luminance part of the video signal is handled separately from the chrominance part and sent in different notifications.

The *temporal* dimension is realized by a layered coding scheme. The first frame in a *gop* (group of pictures) is intra coded, i.e. self contained. The rest of the gop is coded to exploit temporal redundancy. Only the difference between two blocks is sent, taking the layering into consideration. Each additional temporal layer, doubles the frame rate.

The maximum number of notifications generated pr. frame is given by: #superblocks $\times$ #color layers $\times$ #quality layers. A 384×288 pixel frame and a superblock size of 8×6 macroblocks implies 3×3 superblocks, hence a maximum of $9 \times 2 \times 4 = 72$ notifications/frame.

3.2 Description of Demonstration

This technical demonstration will show how different video clients may subscribe to video data with fine grained selectivity. An experimental video client controller is shown in Figure 2, illustrating a subscription and some statistics obtained during execution. Figure 3 shows how e.g. four different video clients may request different parts of the video data (if unclear in the printed copy, please refer to the electronic version) - *upper left*: full quality and colors, *lower left*: luminance and lowest quality for some superblocks, *upper right*: luminance, chrominance, and both for different columns, *lower right*: luminance and lowest quality, except for a part in the middle having full quality and colors.

The content-based networking is handled by Siena [1], extended with IP multicast support [4]. The demonstration will also show how the mapping from the "notification space" to IP multicast addresses may be changed during runtime.

Our Java implementation is capable of encoding/decoding a 384×288 pixel video at 25 frames/sec. on a standard PC.

4. CONCLUSION AND FURTHER WORK

Exploiting content-based networking for video streaming has a potential for offering each video receiver with fine grained selectivity along different quality dimensions, while providing efficient delivery. We currently investigate how to adapt to available resources by e.g. changing subscriptions.

5. REFERENCES

[1] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[2] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A Routing Scheme for Content-Based Networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[3] V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne. Supporting Timeliness and Accuracy in Real-time Content-based Video Analysis. In *Proceedings of the 11th ACM International Conference on Multimedia, ACM MM'03, Berkeley, California, USA*, pages 21–32, November 2003.

[4] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Extending Content-based Publish/Subscribe Systems with Multicast Support. Technical Report 2003-03, Simula Research Laboratory, July 2003.

[5] W. Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):301–317, March 2001.

[6] J. Liu, B. Li, and Y.-Q. Zhang. Adaptive Video Multicast over the Internet. *IEEE Multimedia*, 10(1):22–33, Jan-Mar 2003.

[7] S. McCanne, M. Vetterli, and V. Jacobson. Low-Complexity Video Coding for Receiver-Driven Layered Multicast. *IEEE Journal of Selected Areas in Communications*, 15(6):983–1001, August 1997.

Paper VII

Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming

Viktor S. Wold Eide, Frank Eliassen, and Jørgen Andreas Michaelsen

Published: In Proceedings of the 12th Annual Conference on Multimedia Computing and Networking (MMCN '05), SPIE, San Jose, California, USA, January 2005.

Evaluation: Approximately 100 papers were submitted to MMCN'05. The paper was reviewed by three individuals. As a result of the review process, 16 full papers and 8 short papers were accepted for publication.

Author Contribution: An initial version of the software described in this paper was written by Eide, while the ideas were discussed with Eliassen. However, this implementation was quite rudimentary and did, e.g., not include any code for compression. Michaelsen and Eide collaboratively developed the software into the state described in this paper, both with respect to design and implementation[4]. The decision about which experiments to conduct were decided by Eide and Michaelsen, while Michaelsen did the final measurements. Eide was the driving force behind writing this paper, while both coauthors contributed by commenting on draft versions of the paper.

[4] A history file included in the software, tries to document changes and contributions

Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming

Viktor S. Wold Eide[a,b] Frank Eliassen[a] Jørgen Andreas Michaelsen[b]

[a]Simula Research Laboratory, P.O. Box 134, N-1325 Lysaker, Norway
[b]University of Oslo, P.O. Box 1080 Blindern, N-0314 Oslo, Norway

ABSTRACT

Efficient delivery of video data over computer networks has been studied extensively for decades. Still, multi-receiver video delivery represents a challenge. The challenge is complicated by heterogeneity in network availability, end node capabilities, and receiver preferences.

This paper demonstrates that content-based networking is a promising technology for efficient multi-receiver video streaming. The contribution of this work is the bridging of content-based networking with techniques from the fields of video compression and streaming. In the presented approach, each video receiver is provided with fine grained selectivity along different video quality dimensions, such as region of interest, signal to noise ratio, colors, and temporal resolution. Efficient delivery, in terms of network utilization and end node processing requirements, is maintained. A prototype is implemented in the Java programming language and the software is available as open source. Experimental results are presented which demonstrate the feasibility of our approach.

Keywords: Content-based networking, distributed content-based publish subscribe systems, fine granularity video streaming, scalable video coding, layered video, multi-receiver video streaming

1. INTRODUCTION

Efficient delivery of video data over computer networks, such as the Internet, has been studied extensively for decades. Still, multi-receiver video delivery represents a challenge [1]. Unicast delivery, where clients connect directly to a server, may provide each client with a customized stream, but is inefficient and does not scale. Multicast delivery, provided at the network level or by overlay networks, may improve network efficiency. However, a single multicast stream provides clients with no selectivity. Simulcast delivery may provide clients with a choice between a few streams, each having a different tradeoff between quality characteristics and resource requirements. A combination of layered coding and multicast, as described in [2], is also rather course grained, but improves network efficiency as the amount of redundant information in different streams is reduced. Clearly, the challenge is to provide each video client with *fine grained* and *independent selectivity* along different video quality dimensions, while maintaining efficiency in terms of network and processing resource consumption.

Fine grained and independent selectivity would give each video receiver more freedom when trading off video quality in different dimensions against the available resources, such as network bandwidth, processing capabilities, display resolution, and power. A video receiver may then prefer to increase the temporal resolution, while another video receiver may prefer to increase the signal to noise ratio, without any conflicts. Fine grained selectivity with respect to regions of interest may also support virtual pan and zoom in very high quality video streams. Maybe even more challenging is adapting quality in regions based on visual focus - high quality for the regions looked at and less quality in the periphery. For domains such as real-time distributed video content analysis, timeliness and accuracy requirements may necessitate parallel processing, as described in [3]. Parallel processing of video data also represents a heterogeneity challenge. Workload may be distributed among different processing entities by partitioning the video data spatially and/or temporally. The above reasoning illustrates that fine grained and

Further author information:
V.S.W.E.: E-mail: viktore@simula.no, viktore@ifi.uio.no
F.E.: E-mail: frank@simula.no
J.A.M.: E-mail: jorgenam@ifi.uio.no

independent selectivity may support heterogeneity, where receivers range from small mobile hand held devices to high resolution display walls, as well as parallel processing.

In [4], an extension for the CORBA Event Service is described which supports stream events and multicast delivery of video data. However, in channel based systems each notification is not forwarded (routed) independently.

In this paper we demonstrate that content-based networking is a promising technology which enables fine grained and independent selectivity. Content-based networking provides rich routing capabilities as well as a level of indirection. This allows a more dynamic and flexible video streaming solution compared to more traditional approaches, such as layered multicast. In the presented approach, a single video server may handle a large number of heterogeneous video receivers. Each video receiver may independently select region of interest, signal to noise ratio, colors, and temporal resolution. The contribution of this work is the bridging of content-based networking with well known techniques from the fields of video compression and streaming. A prototype has been implemented as a proof of concept [5]. Performance measurements indicate that efficient delivery is maintained, in terms of bit rates and end node processing requirements.

The video streaming approach presented in this paper is developed in the context of the DMJ (Distributed Media Journaling) project [6]. The project develops a framework targeting the application domain of distributed real-time video content analysis. The video streaming software is also used in our research related to this application domain.

The rest of the paper is structured as follows. First, in Sect. 2 we present some background information on content-based networking. Then, in Sect. 3 we describe our architecture for video streaming over content-based networking. In particular, the different techniques used to achieve fine grained selectivity along each video quality dimension are described. A prototype implementation is presented in Sect. 4, along with performance measurements for coding efficiency and processing requirements. In Sect. 5 we conclude and describe further work.

2. CONTENT-BASED NETWORKING

In this section, some background information regarding content-based networking and its relation to distributed content-based publish subscribe systems is described. For a survey on the publish subscribe communication paradigm and the relations to other interaction paradigms, the reader is referred to [7].

In content-based networking, as described in [8], messages are forwarded based on content, and not on an explicit address. Each message contains a set of attribute/value pairs and clients express interest in certain messages by specifying predicates over these attributes and values. The predicates are used by the network nodes for routing. Messages are forwarded based on content-based routing tables and delivered only to clients having matching selection predicates. Filtering of messages is pushed towards the source while replication is pushed towards the destinations. Consequently, each message should traverse a link at most once. Content-based networks are currently realized as overlay networks.

Distributed content-based publish subscribe systems are intimately related to content-based networking. Examples of content-based publish subscribe systems include Elvin [9], Gryphon [10], Hermes [11], and Siena [12]. In such systems, the messages are called event notifications, or just notifications for short. Clients may inject notifications into the network by publishing, as illustrated in Fig. 1. Other clients express their interests in subscriptions, as predicates over the attribute/value pairs, and are notified accordingly. In a video streaming scenario, each video receiver subscribes and thereby expresses interest in some part of the video signal.

As an example, consider a scheme for video streaming which supports region of interest selectivity by dividing each frame in $m \times n$ regions. In this scheme, each notification contains four attribute names - *sid* (stream identifier), *row* (row), *col* (column), and *blob* (binary video data). A video server may then publish each video frame as $m \times n$ notifications. In each notification values are assigned to each attribute. As an example, a notification may have the following content: [sid=10 col=1 row=2 blob=q34i23QR...D]. A video receiver may then express interest in only one of the columns with the following subscription: [sid=10 col=1]. A second video receiver may receive only one of the rows using another subscription, e.g.: [sid=10 row=2]. Similarly, a third video receiver may receive all regions by using a subscription such as: [sid=10].

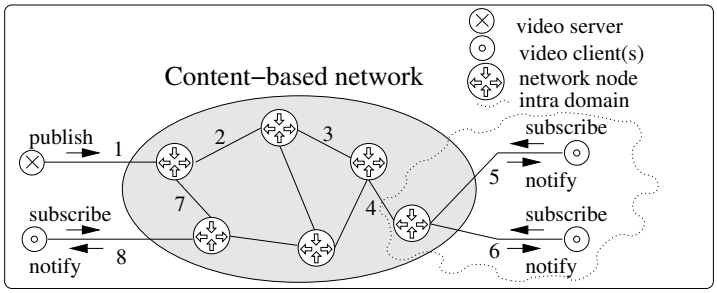

Figure 1. Content-based networking example.

Both subscriptions and notifications are being pruned *inside* the content-based network. As an example, consider the case where two video clients are connected to the same network node, as illustrated in Fig. 1. Notifications are not forwarded in the network before at least one video receiver has expressed interest. When the first video receiver subscribes and thereby registers interest in some part of the video signal, the subscription may get forwarded over the links numbered $5, 4, 3, 2$, and 1. State is then maintained in the network nodes to allow notifications to flow e.g. on the reverse path. When the client connected by the link numbered 6 subscribes, the first network node only forwards the subscription if this new subscription is not covered by the first subscription. Consequently, both subscriptions and notifications are pruned in the network.

Different protocols may be used concurrently for different links in the content-based network, such as TCP for some server-server links and UDP over IP multicast for other server-server links.

The architectures and algorithms for scalable wide area publish subscribe systems have been studied extensively [10, 12]. The principles and techniques used for routing and forwarding notifications between servers in distributed content-based publish subscribe systems are similar to those used by IP routers in order to support IP multicast. In [13], an efficient multicast protocol for content-based publish subscribe systems is presented. The challenge of utilizing *native* multicast support for content-based publish subscribe systems is well known [7]. Simulation results for some algorithms for distributing notifications are presented in [10], where the network nodes are treated as the communication endpoints. Complementary to the WAN case is the challenge of efficiently distributing very high rate event notifications between a large number of clients within a smaller region, e.g. a LAN or an administrative domain. In Fig. 1 this corresponds to the notifications forwarded over the links numbered $5, 6$, and to a potentially large number of other interested clients within the domain. In Sect. 4 performance issues and measurements are presented which show that content-based publish subscribe systems may handle the rates required for streaming compressed high quality video.

3. FINE GRANULARITY VIDEO STREAMING

The different techniques we use to achieve fine granularity selectivity along each video quality dimension is described in this section. Different video coding techniques were considered in the design, but the judgements were based on the suitability with respect to content-based networking and similarities with MPEG. In other words, the techniques used are well known in the fields of video compression and streaming.

Each video server connects to the content-based network to publish notifications. Each notification encapsulates encoded video data and contains a subset of the video data for a single frame. The content-based network is then responsible for forwarding and delivering each notification to video clients having matching subscriptions.

The purpose of the coding and encapsulation scheme presented in this section is to allow each video receiver to independently tradeoff between the video quality characteristics and the resource requirements. The coding scheme supports selectivity along the following video quality dimensions: *region of interest, signal to noise ratio, colors*, and *temporal resolution*. In the following we describe the techniques used to achieve selectivity for each of these dimensions.

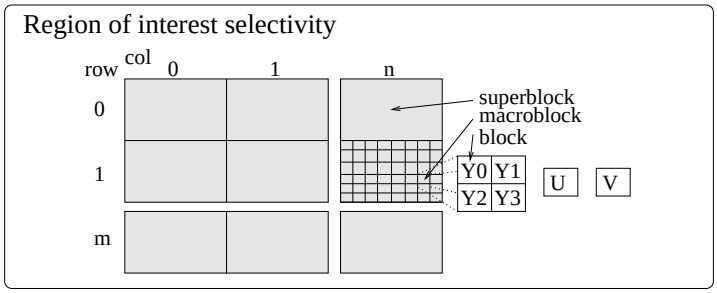

Figure 2. Region of interest selectivity scheme.

3.1. Region of Interest Selectivity

Video receivers may select the *region of interest* in terms of so-called superblocks. A superblock contains a number of 16×16 pixel macroblocks. Each macroblock contains luminance and chrominance blocks. This is illustrated in Fig. 2. A superblock represents the smallest selectable *region* and is self contained. Superblocks are indexed by row and a column number and the attribute names are *row* and *col* respectively. This implies that if a video receiver is interested in a region which is located within a superblock, the client must receive video data for the whole superblock and discard the uninteresting parts. Given that each superblock is self contained, motion compensation must be restricted to within such superblocks. However, each superblock may be analyzed by a separate computer in a distributed video content analysis application.

3.2. Signal to Noise Ratio Selectivity

In the *signal to noise ratio* (SNR) dimension, a layered coding is used. Each block within a macroblock is transformed to the frequency domain by using the Discrete Cosine Transform (DCT). The DCT coefficients are quantized and viewed as bit-planes. These bit-planes have certain statistical properties which are exploited by variable length coding. The different bit-planes are mapped to different SNR layers, as illustrated in Fig. 3. Notifications for the base layer contain the most significant bits. The sign bit for a DCT value is encoded together with its most significant bit. The attribute for selecting the signal to noise ratio is named *ql* (quality). The reader is referred to [14] for an in-depth description of bit-plane coding for Fine Granularity Scalability in MPEG-4.

Currently, neither the quantization nor the partitioning of the bit planes into SNR layers are varied during a video streaming session. However, we consider the alternative approach of allocating a fixed number of bits to the different SNR layers, by varying the quantization and/or the partitioning into bit planes.

3.3. Luminance and Chrominance Selectivity

With respect to *colors*, the luminance part of the video signal is handled separately from the chrominance part. The video encoder takes as input the video signal represented in YUV420 planar format. In other words, the chrominance part of the signal is sub-sampled 2:1 both horizontally and vertically. Hence, each 16×16 pixel macroblock consists of four luminance 8×8 blocks, a 8×8 U block, and a 8×8 V block.

Currently, the video server partitions the luminance part (Y) and the two chrominance parts (U and V) of the video signal in the same number of SNR layers. The encoded luminance part of the video signal is sent in separate notifications, while each notification carrying color information encapsulate both U and V. The motivation for this approach was to increase the payload in each notification and thereby reduce the total number of notifications generated. The number of notifications could have been kept the same by encapsulating U and V in different notifications and halving the number of SNR layers for the U and the V components. This illustrates a tradeoff where the expected usage is taken into account, i.e. that video receivers usually are interested in either both color components or none.

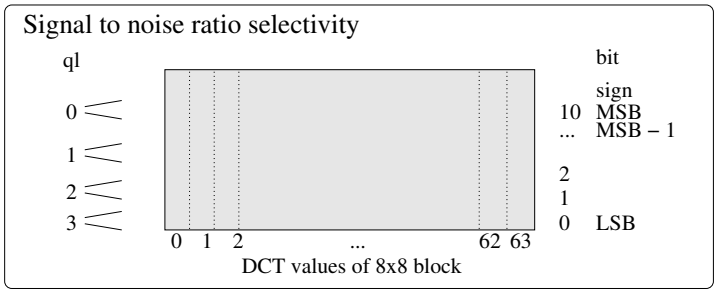

Figure 3. Signal to noise ratio selectivity scheme.

The attribute for selecting luminance and chrominance is named f (color flag). As an example of the fine granularity selectivity, a receiver may subscribe to the luminance part of the video signal at a higher SNR than the chrominance part, or visa versa. Currently, this requires two subscriptions, e.g. [f=0 ql<=3] and [f=1 ql<=1]. This could also be done by having a scheme with "quality luminance" and "quality chrominance" attributes. Then only a single subscription would have sufficed. However, the important point is that each video receiver may independently tradeoff luminance and chrominance according to its own preferences.

3.4. Temporal Resolution Selectivity

The *temporal* dimension is also realized by a layered coding scheme. The first frame in a *group of pictures* (GOP) is intra coded and thus self contained. The rest of the GOP is coded to exploit temporal redundancy. Each additional temporal layer increases the frame rate.

The currently implemented scheme is illustrated in Fig. 4. Each additional temporal layer adds a frame in between all frames at lower layers. Consequently, the frame rate is doubled when adding a temporal layer. Additionally, irrespectively of the number of layers a client requests, frames are spread out evenly in time. The first frame in a GOP, with index 0, is self contained. The other frames are predictively coded, similar to P frames in MPEG. The reference frame used for calculating the difference is the previous frame at a lower layer. Only the difference between a block in the reference frame and a block in the current frame is coded. The attribute for selecting the temporal resolution is named *tl* (temporal layer).

An advantage of such a scheme is that the end to end delay can be kept low, because the sender may encode a frame as soon as it becomes available. Only (some of the) previously encoded frames are used when encoding a new frame.

The amount of redundant information is small in the two lowest temporal layers. But when adding more layers, the amount of redundant information may increase. As an example, consider the case where the content of a block changes between GOP index 0 and 1, but then remains the same for the rest of the GOP. First the block difference for the frames with GOP index 0 and 1 is encoded and sent at layer 3. Then the frame with GOP index 0 is again used as reference when encoding the frame with GOP index 2. This is necessary, because some video clients may receive only notifications for temporal layer 2 and below. Consequently, the information sent at layer 2 for this block is the same as the information sent previously at layer 3. The same will then happen for the frame with GOP index 4, which is sent at layer 1.

We are currently working on a scheme which reduces the bit rate at the cost of an increased end to end delay. The amount of redundant information sent at the higher layers can be reduced by allowing references to future frames, similar to B frames in MPEG. By following the example above, the block can then be encoded in the frame with GOP 4. Then the frames with GOP indexes 1, 2, and 3 could be encoded very efficiently relatively the frame with GOP index 4. Such out of ordering encoding increases the end to end delay and requires more buffering and computational resources.

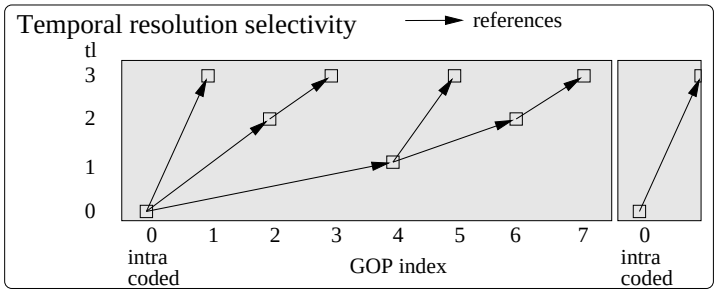

Figure 4. Temporal resolution selectivity scheme.

3.5. Fine Grained Selectivity and Network Efficiency

With typical network equipment each packet has a high initial cost and an additional cost per byte of payload. Therefore, in order to provide efficient network utilization for high volume video data, each packet should carry a reasonable amount of the video information. On the other hand, in order to provide fine granularity selectivity and a good match between client interests and what is actually received over the network, each notification should contain little information. Clearly, the challenge is to balance selectivity against efficient network delivery.

The number of superblocks, SNR layers, and color layers determine the tradeoff between selectivity and the number of notifications generated. The maximum number of notifications generated per frame in our coding scheme is given by: #superblocks $\times$ #color layers $\times$ #SNR layers. Hence, the granularity may be customized in order to suit the requirements imposed by a given application. As an example, a CIF sized video frame (352×288 pixels) and a superblock size of 11×6 macroblocks implies three superblock rows and two superblock columns. With four SNR layers and two color layers, the maximum number of notifications per frame is $6 \times 2 \times 4 = 48$. In the following section, this level of granularity has been used for performance measurements.

4. PROTOTYPE

In the following we present our implementation for video streaming over content-based networking. First some information regarding content-based networking is given. Then we present the video streaming software and performance numbers related to coding efficiency and processing requirements.

4.1. Content-Based Networking Software

For content-based networking we have used Siena [12], a distributed content-based publish subscribe system. In the DMJ project, we have made some extensions to the Siena software. The motivation for these extensions was to support applications requiring high notification rates in combination with a large number of interested receivers within a smaller region, such as a LAN or an administrative domain. The architecture of our distributed content-based event notification service is described in [15]. The service takes advantage of the available performance and native multicast support provided by current "off the shelf" network equipment. In short, a mapping from the "event notification space" to IP multicast addresses is specified. Each notification is mapped to an IP multicast address and efficiently forwarded to all clients having matching subscriptions. The service is designed for use within a LAN or an administrative domain. A client may publish several thousand notifications, carrying several MBytes of data, per second. The service is unaffected by the number of interested parties, due to the use of native network level and link level multicast. Such rates are more than sufficient for streaming compressed high quality video.

It should be noted that Siena in its current implementation uses a text based format, which *may* incur a substantial overhead when transferring compressed binary video data. In the following, we were therefore restricted to measure data rates as seen by the video receivers. A more efficient implementation of encapsulation in Siena is future work.

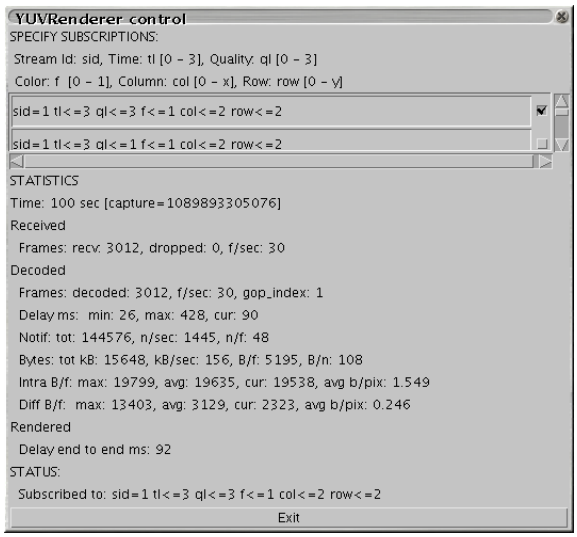

Figure 5. An experimental video client controller.

4.2. Video Streaming Software

The presented video streaming software exploits well known video compression techniques, but is implemented from scratch because other MPEG implementations seemed unsuited. The software is implemented in the Java programming language. Different versions of Java, J2SE 1.4 and 1.5 from Sun Microsystems, have been used and tested. The software is open source and is available from our project web page [6].

Fig. 5 shows a screenshot of an experimental video receiver controller. The controller is used to interactively modify subscriptions and to subscribe and unsubscribe to video data. During execution, the controller displays some statistics about the received, decoded, and rendered video data.

Fig. 6 illustrates that different video receivers may independently select different parts of the video signal. (If the images are unclear in the printed copy, please refer to the electronic version.) The figure illustrates the effect of selecting full quality and colors (upper left), only luminance, only color and both (upper right), only luminance and some superblocks at lowest quality (lower left), and only luminance and low quality, except for a region having full quality and colors (lower right). A video receiver may also specify different temporal resolutions for different superblocks within a frame.

4.3. Video Streaming Performance Measurements

In this section we present performance measurements for coding efficiency and processing requirements for the supported video quality dimensions, i.e. *region of interest, signal to noise ratio, colors*, and *temporal resolution*. The effect of varying the quantization scale was also measured. Ideally, the bit rates and processing requirements should drop when a video receiver reduces the quality in any dimension. All measurements were done at the video receiver side.

With respect to the bit rates, only the amount of video data was measured. The protocol overhead associated with encoding the attribute names and the other attribute values in each notification is not measured. The overhead depends on the length of the attribute names and the encoding of the values. In addition to the attributes already described, the video streaming software internally uses a sequence number attribute, a timestamp attribute, and an attribute for the binary compressed video data. All attribute values are integers, except for the video data which is encoded in a byte array.

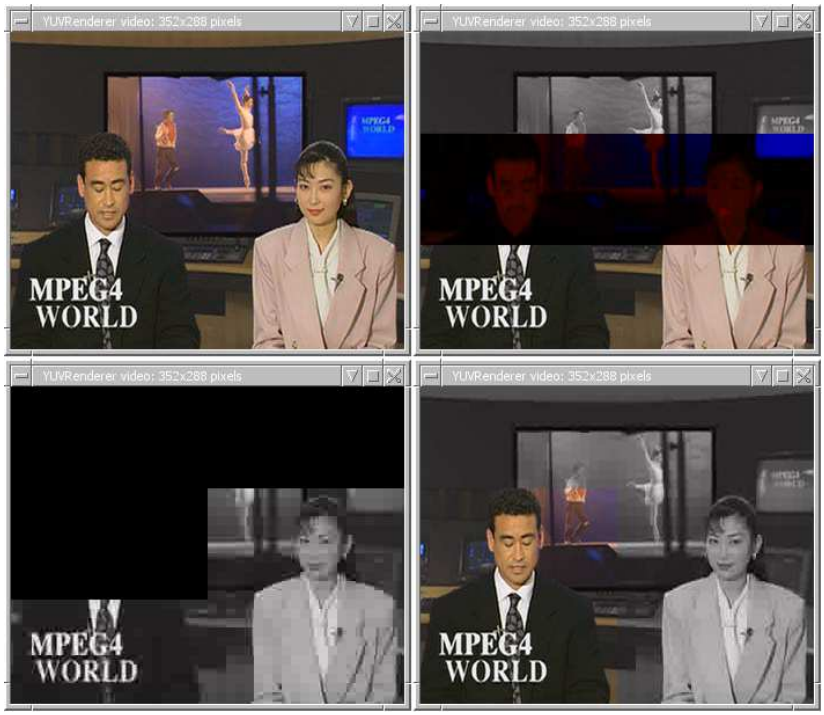

Figure 6. Four video clients with different interests.

With respect to processing requirements, a single thread was responsible for decoding the video data received in notifications. The percentage of time (user and system) spent by this thread was measured by instrumenting the software with native code using the *times* system call. The numbers gathered were consistent with the numbers reported by programs such as *top* and *xosview*.

For the measurements, we used some well known video test sequences. These sequences are *News* and *Foreman* in CIF resolution (352 × 288 pixels). The full frame rate in all experiments was 30 frames pr. second. The News sequence has steady camera and little motion, while the Foreman sequence has camera motion. Standard dual AMD Athlon MP 2400+ PCs running Debian GNU/Linux were used. The PCs were connected by 100Mbps switched Ethernet. Java Standard Edition build 1.4.2_03-b02 was used for compilation and execution.

4.3.1. DCT Coefficient Quantization

In this experiment the compression performance for varying values of the *quantizer scale* (qscale) was measured. The purpose was to fix the qscale value for the other experiments. The qscale value determines the quantization of the DCT coefficients. The video receiver subscribed to the full temporal resolution, all quality layers, all superblocks, but only the luminance part of the video signal. The decompressed video was written to a file and compared to the video signal provided to the encoder. The *peak signal to noise ratio* (PSNR) was calculated, based on the *mean square error*. The average number of *bytes pr. frame* and the number of *bits pr. pixel* (bpp) for both the intra frames and the difference frames were measured. The results are listed in Table 1. The table presents the number of *kilo bits pr. second* (kbps). A qscale value of four gives relatively good PSNR for both the News and the Forman sequence. For the Foreman sequence and low qscale values, the number of bits pr. pixel for the difference coded frames are similar to those for the intra coded frames. The reason is that motion compensation is not implemented in the current prototype.

Based on these observations, a qscale value of four and only the News sequence is used in the rest of the experiments.

Table 1. DCT coefficient quantization measurements.

News CIF Y				
qscale	PSNR	bpp intra	bpp diff	kbps
0	46.46	4.15	0.69	3495
1	44.47	2.42	0.49	2275
4	39.32	1.18	0.20	1003
8	35.50	0.82	0.11	621
16	31.77	0.56	0.05	366
32	28.38	0.36	0.02	207
Foreman CIF Y				
qscale	PSNR	bpp intra	bpp diff	kbps
0	47.99	4.87	3.98	12456
1	43.38	2.88	2.93	8893
4	36.25	1.33	1.06	3339
8	32.72	0.88	0.51	1703
16	29.53	0.58	0.22	819
32	26.57	0.39	0.08	377

Table 2. Region of interest measurements.

News CIF Y					
col	row	% CPU	bpp intra	bpp diff	kbps
0	0	9.50	0.70	0.16	116
0	1	11.82	1.20	0.29	208
0	2	5.79	1.52	0.03	114
1	0	7.08	0.88	0.20	147
1	1	11.99	1.34	0.40	268
1	2	8.17	1.44	0.12	149
Sum		54.35	1.18	0.20	1002
All		55.37	1.18	0.20	1003

4.3.2. Region of Interest Selectivity

The purpose of this experiment was to validate that each superblock is encoded independently. Hence, the data rates and processing requirements should drop accordingly when only some superblocks are received and decoded. Only the luminance part of the video signal was received, but all SNR layers and the full temporal resolution.

First the coding efficiency for each superblock was measured by changing the values for the row and the column attributes in the subscriptions. The results are listed in Table 2. The table illustrates that both the processing requirements and the bit rates are reduced when only a single superblocks is received. The processing requirements and bit rates vary somewhat between the different superblocks due to differences in the video content from superblock to superblock. In the second to last row the numbers for all superblocks have been added or averaged. The last row contains the measured numbers when subscribing to all superblocks at once. The numbers are almost identical.

We also measured the processing requirements as a function of the number of superblocks decoded. The numbers are presented in Fig. 7. The measured values as well as the average values are plotted. E.g., in the case when four superblocks were received, the CPU requirements were measured for different combinations of superblocks in order to reduce the effect of differences in media content for different superblocks. The dashed line represents the linear approximation and indicates that the processing requirements are linearly in the number of superblocks decoded.

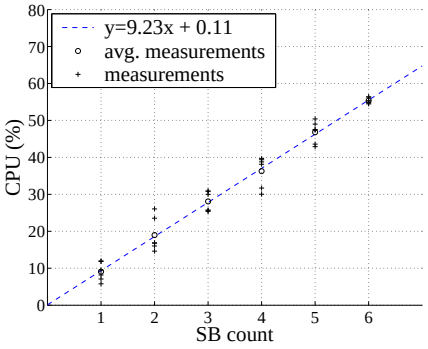

Figure 7. Region of interest measurements.

Table 3. Signal to noise ratio measurements.

News CIF Y					
ql	PSNR	% CPU	bpp intra	bpp diff	kbps
+ 3	39.32	55.37	1.18	0.20	1003
+ 2	34.72	54.20	0.72	0.11	592
+ 1	30.24	53.31	0.46	0.06	349
base	26.63	53.10	0.29	0.03	213

These two experiments show that each superblock is handled individually and that processing and bandwidth consumption scale according to the regions of interest.

4.3.3. Signal to Noise Ratio Selectivity

The purpose of this experiment was to measure the bit rates at the different quality layers. The full temporal resolution, but only the luminance part of the video signal was received. The number of quality layers was reduced by one, until the video receiver only received the base layer. The results are listed in Table 3. From the table, it can be seen that the decoding processing requirements are not significantly reduced when reducing the number of SNR layers.

From the table it can be seen that the PSNR decreases as the number of SNR layers are reduced. The required bit rates also decreases. By comparing the numbers in this table with the numbers in Table 1, it can be seen that reducing the number of SNR layers is similar to increasing the qscale value.

4.3.4. Luminance and Chrominance Selectivity

In this experiment, the bit rates and processing requirements were measured for receiving luminance and chrominance information separately and combined. The full temporal resolution and all SNR layers were received. The measured results are presented in Table 4. From the table it can be seen that both processing and bit rates are reduced when only luminance or chrominance video data is received. The numbers for both bit rates and processing requirements add up. This shows that the luminance and the chrominance part of the video signal are handled separately.

4.3.5. Temporal Resolution Selectivity

The effect of reducing the frame rate by subscribing to a reduced number of temporal layers was measured in this experiment. All superblocks were received in full quality, but only the luminance part. The measured results are presented in Table 5. The full temporal resolution was 30 frames pr. second, while the base layer represents 3.75 frames pr. second. The table shows that both processing and bit rates are reduced when the number of temporal layers is reduced, although not proportionally. At the base layer, only the intra coded frames are received. Intra

Table 4. Luminance and chrominance measurements.

News CIF				
f	% CPU	bpp intra	bpp diff	kbps
YUV	78.00	1.55	0.24	1257
UV	23.99	0.37	0.04	253
Y	55.37	1.18	0.20	1003

Table 5. Temporal resolution measurements.

News CIF Y				
tl	% CPU	bpp intra	bpp diff	kbps
+ 3	55.37	1.18	0.20	1003
+ 2	38.33	1.18	0.27	774
+ 1	26.40	1.18	0.33	586
base	19.34	1.18	0.00	448

coded frames are more demanding in terms of both processing and bit rate requirements. The average number of bits per pixel for the difference frames increases as the number of temporal layers is reduced. The reason is most likely that the time interval between frames increases as the number of temporal layers is reduced, and therefore the difference between the frames is also bigger.

5. CONCLUSION AND FURTHER WORK

In this paper, we have presented an approach for exploiting content-based networking for video streaming. The novel feature is that each video receiver is provided with fine grained selectivity along different video quality dimensions, while efficient delivery, in terms of bit rates and end node processing requirements, is maintained. The video quality dimensions considered in this paper are region of interest, signal to noise ratio, colors, and temporal resolution. A prototype is implemented as a proof of concept. Performance measurements show that bandwidth and processing requirements decrease significantly when video quality is reduced in the different dimensions. The software is available as open source.

We are currently working on motion compensation, adaptation, and spatial scalability support. With respect to adaptation, video receivers may change their subscriptions in order to reduce the resource requirements. Additionally, the content-based network itself may perform adaptation. Closely related to resource availability is Quality of Service issues, which is considered interesting in the content-based networking research community. We are also considering closer integration with MPEG-4, e.g., each superblock may be encoded as a video object plane. The coding scheme, using attribute names and values, may also be suitable for storage in database systems.

ACKNOWLEDGMENTS

We would like to thank all persons involved in the Distributed Media Journaling project for contributing to the ideas presented in this paper. Additionally, Hans Ole Rafaelsen took part in valuable discussions and commented on both the ideas and the implementation issues. We also would like to thank the reviewers for valuable feedback.

The DMJ project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431.

REFERENCES

1. J. Liu, B. Li, and Y.-Q. Zhang, "Adaptive Video Multicast over the Internet," *IEEE Multimedia* **10**, pp. 22–33, Jan-Mar 2003.
2. S. McCanne, M. Vetterli, and V. Jacobson, "Low-Complexity Video Coding for Receiver-Driven Layered Multicast," *IEEE Journal of Selected Areas in Communications* **15**, pp. 983–1001, August 1997.
3. V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne, "Supporting Timeliness and Accuracy in Real-time Content-based Video Analysis," in *Proceedings of the 11th ACM International Conference on Multimedia, ACM MM'03, Berkeley, California, USA*, pp. 21–32, November 2003.
4. D. Chambers, G. Lyons, and J. Duggan, "Stream Enhancements for the CORBA Event Service," in *Proceedings of the ACM Multimedia (SIGMM) Conference, Ottawa*, pp. 61–69, October 2001.
5. V. S. W. Eide, F. Eliassen, and J. A. Michaelsen, "Exploiting Content-Based Networking for Video Streaming," in *Proceedings of the 12th ACM International Conference on Multimedia, Technical Demonstration, ACM MM'04, New York, New York, USA*, pp. 164–165, October 2004.
6. "The Distributed Media Journaling Project." http://www.ifi.uio.no/~dmj/.
7. P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys (CSUR)* **35**, pp. 114–131, June 2003.
8. A. Carzaniga, M. J. Rutherford, and A. L. Wolf, "A Routing Scheme for Content-Based Networking," in *Proceedings of IEEE INFOCOM 2004*, (Hong Kong, China), March 2004.
9. B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, "Content Based Routing with Elvin4," in *Proceedings of AUUG2K, Canberra, Australia*, June 2000.
10. L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman, "Exploiting IP Multicast in Content-Based Publish-Subscribe Systems," in *Proceedings of Middleware 2000*, *LNCS 1795*, pp. 185–207, Springer-Verlag, 2000.
11. P. R. Pietzuch and J. M. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," in *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pp. 611–618, IEEE Computer Society, July 2002.
12. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Transactions on Computer Systems* **19**, pp. 332–383, August 2001.
13. G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," in *Proceedings of ICDCS*, pp. 262–272, IEEE, 1999.
14. W. Li, "Overview of Fine Granularity Scalability in MPEG-4 Video Standard," *IEEE Transactions on Circuits and Systems for Video Technology* **11**, pp. 301–317, March 2001.
15. V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo, "Extending Content-based Publish/Subscribe Systems with Multicast Support," Tech. Rep. 2003-03, Simula Research Laboratory, July 2003.

Paper VIII

Real-time Video Content Analysis: QoS-Aware Application Composition and Parallel Processing

Viktor S. Wold Eide, Ole-Christoffer Granmo[5],
Frank Eliassen, and Jørgen Andreas Michaelsen

Published: Submitted to ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), April 2005.

Author Contribution: This is a project article which extends Paper V and integrates the results presented in Paper IV, VI, and VII. Additionally, the article includes a new prototype application which is based on the software described in these earlier papers. Hence, the contributions of Eide follow from the contributions in Paper IV, V, VI, and VII. The necessary additional implementation work as well as the experimental measurements were done collaboratively by Granmo, Michaelsen, and Eide. The conceptual integration was done collaboratively by all authors, while the writing of the paper was done mainly by Granmo and Eide. The other authors commented on draft versions of the paper.

[5]The two first authors are listed alphabetically

Real-time Video Content Analysis: QoS-Aware Application Composition and Parallel Processing

VIKTOR S. WOLD EIDE[1]
Simula Research Laboratory
OLE-CHRISTOFFER GRANMO[1]
Agder University College
FRANK ELIASSEN
Simula Research Laboratory
and
JØRGEN ANDREAS MICHAELSEN
University of Oslo

Real-time content-based access to live video data requires content analysis applications that are able to process video streams in real-time and with an acceptable error rate. Statements as these express quality of service (QoS) requirements. In general, control of the QoS provided can be achieved by sacrificing application quality in one QoS dimension for better quality in another, or by controlling the allocation of processing resources to the application. However, controlling QoS in video content analysis is particularly difficult, not only because main QoS dimensions like accuracy are non-additive, but also because *both* the communication- and the processing resource requirements are challenging.

This paper presents techniques for QoS-aware composition of applications for real-time video content analysis, based on dynamic Bayesian networks. The aim of QoS-aware composition is to determine application deployment configurations which satisfy a given set of QoS requirements. Our approach consists of: (1) an algorithm for QoS-aware selection of configurations of feature extractor and classification algorithms which balances requirements for timeliness and accuracy against available processing resources, (2) a distributed content-based publish/subscribe system which provides application scalability at multiple logical levels of distribution, and (3) scalable solutions for video streaming, filtering/transformation, feature extraction, and classification.

We evaluate our approach based on experiments with an implementation of a real-time motion vector based object-tracking application. The evaluation shows that the application largely behaves as expected when resource availability and selections of configurations of feature extractor and classification algorithms varies. The evaluation also shows that increasing QoS requirements can be met by allocating additional CPUs for parallel processing, with only minor overhead.

[1]The two first authors are listed alphabetically

The Distributed Media Journaling (DMJ) project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431.
Eide, Simula Research Laboratory, P.O. Box 134, N-1325 Lysaker, Norway, viktore@simula.no
Granmo, Agder University College, Grooseveien 36, N-4876 Grimstad, Norway, ole.granmo@hia.no
Eliassen, Simula Research Laboratory, P.O. Box 134, N-1325 Lysaker, Norway, frank@simula.no
Michaelsen, University of Oslo, P.O. Box 1080 Blindern, N-0314 Oslo, Norway, jorgenam@ifi.uio.no

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*Video analysis*

General Terms: Algorithms, design, measurement, performance

Additional Key Words and Phrases: Real-time video content analysis, parallel processing, task graph scheduling, event-based communication, publish/subscribe, QoS and resource management

1. INTRODUCTION

There is an increasing need for applications that can analyse the content of multiple media streams in real-time. Real-time performance is important for interactive applications and feedback control systems, such as smart rooms, automated surveillance systems, and road traffic control systems[Beymer et al. 1997; Chen et al. 2001; Ozer and Wolf 2001].

Real-time content analysis is an active research field where efficient techniques for e.g. multi-object detection and tracking have been found. Pattern classification systems which automatically classify media content in terms of high-level concepts, represent one particular approach. 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. In this context, real-time performance implies that the filtering, feature extraction, and classification of media data is performed at least as fast as the data is made available to the application.

When building a real-time content analysis application, not only must the processing properties of the application be considered, but also the content analysis properties. For instance, misclassifying events when monitoring an airport for security reasons may be more critical than misclassifying events when indexing a baseball video stream. Furthermore, when a classification error occurs, the consequences of various types of errors may have different costs. E.g., it may be more costly to misclassify a friendly airplane as hostile, than to classify an enemy airplane as friendly (depending on the situation).

Application critical properties as those discussed above can be referred to as Quality of Service (QoS) dimensions. In this paper, we consider the QoS dimensions latency, temporal resolution, and accuracy. We view these as the most relevant for the targeted application domain. However, controlling these QoS dimensions of video content analysis applications is difficult, since some of them (e.g., accuracy) do not have strict additive behavior. In contrast, in earlier work on QoS management, it is not unusual to assume additive behavior of QoS dimensions to simplify the reasoning about and control of end-to-end QoS, as in [Gu and Nahrstedt 2005].

In this article we present techniques for QoS-aware composition of applications for real-time video content analysis. QoS-aware application composition is the process of determining the deployment configuration of an application to satisfy a given set of application QoS requirements. First of all, we propose a dynamic Bayesian network (DBN) algorithm for QoS-aware selection of feature extractor- and classification configurations. The algorithm balances requirements of timeliness and

accuracy against available processing resources. Additionally, the algorithm exploits that DBNs allow features to be removed during classification, at the cost of decreased accuracy, but for the benefit of reduced processing requirements (CPU and network). Configurations are assumed deployed in a particular generic application architecture that allows applications to be independently distributed and parallelized at muliple logical levels of processing. In essence, we propose scalable solutions for video streaming, filtering/transformation, feature extraction, and classification. This includes a novel classifier that parallelize inference in dynamic Bayesian networks. Furthermore, a scalable content-based publish/subscribe interaction mechanism supports communication of video and feature data among multiple distributed senders and receivers. The purpose is to allow processing resources to be focused at points in the processing that are not performing well enough to satisfy the required QoS.

The work presented in this article extends the work presented in [Eide et al. 2003], both in terms of architecture and evaluation of our work. In particular we have extended the content-based publish/subscribe service with multicast support [Eide et al. 2003] and implemented video streaming on top of this service [Eide et al. 2004; 2005]. Furthermore, we provide significantly extended empirical results based on observations of a real video content analysis application that integrates the use of the newly developed content-based publish/subscribe service throughout the whole application architecture. This way we are able to observe if the QoS properties of the outputs of the application are as expected. The new experiments show that the bottlenecks observed in [Eide et al. 2003] do not occur with the use of the new video streaming service.

The rest of this paper is structured as follows: First, in Section 2, we present a generic architecture for (video) content analysis applications, and a QoS model for such applications. We also discuss architectural requirements that are important for supporting QoS-aware composition. Based on these requirements, details of our approach to QoS-aware composition are presented in Section 3. Additionally, we propose scalable solutions for communication, streaming, filtering/transformation, feature extraction, and classification. In Section 4, we present evaluation results, while section 5 compares our approach with representative previous work. Lastly, in Section 6, we conclude and provide some outlook to further work.

2. GENERIC APPLICATION ARCHITECTURE

Our proposed solution to QoS-aware application composition assumes a particular generic application architecture that allow applications to be independently distributed and parallelized at multiple logical levels. This section covers the application architecture, the QoS model, and the architectural requirements important for supporting QoS-aware composition of video content analysis applications.

2.1 Generic Content Analysis Architecture

A general approach for building content analysis applications is to combine low-level quantitative video processing into high-level concept recognition. Typically, such applications are logically organized as a hierarchy of modules, each encapsulating a video processing task, as illustrated in Figure 1. A task is a well-defined algorithm involved in the video analysis process.

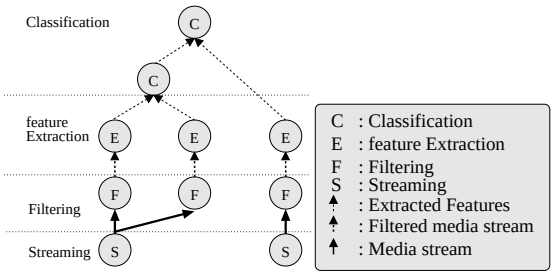

Fig. 1. Content analysis hierarchy example.

We define task categories according to their functionality in the system. At the lowest level of the hierarchy there are tasks representing video streaming sources. At the level above, the video streams are filtered and transformed by filtering tasks. The transformed video streams are then fed to feature extraction tasks as video segments (e.g., video frame regions). Feature extraction tasks operate on the video segments from the transformed video streams, and in the case of a video frame region, calculate color, texture, shape, and motion characteristics. Finally, results from feature extraction tasks are reported to classification tasks higher up in the hierarchy that are responsible for detecting high level domain concepts, such as a moving object in a video stream. In other words, classification is interpretation of extracted features in some application specific context. We will hereafter denote the streaming, filtering, feature extraction, and classification by the letters S, F, E, and C respectively, as seen in Figure 1.

Tasks generally form a directed acyclic graph where the tasks are represented by the nodes in the graph, and the edges represent the directed flows of data between tasks. We refer to a task graph specifying the architecture of a video content analysis application as the *composition specification* of the application.

Often, the above type of content analysis applications are implemented as monolithic applications making reuse, development, maintenance, and extensions by third parties difficult. Such applications are often executed in a single process, unable to benefit from distributed processing environments. However, it should be possible to deploy the tasks of a task graph to different processors within a computer and/or to different interconnected computers. In this case the edges are mapped to network level bindings or to intra-host communication mechanisms, depending on the location of communicating tasks. We discuss this further below.

2.2 Application QoS Model

In this article we use to the term QoS to refer only to the timeliness and accuracy of the outputs of the application. The QoS model therefore includes the following QoS dimensions: *accuracy*, *temporal resolution*, and *latency*.

The *accuracy* of a media content analysis application can be characterized by its estimated error rate, defined as the number of misclassifications divided by the total number of classifications when analysing a set of media streams. Depending on the

media content analysis application, various levels of error rate may be acceptable. E.g., misclassifying events when monitoring an airport for security reasons may be more critical than misclassifying events when indexing a baseball video stream.

The *temporal resolution* dimension specifies the minimum temporal length of an event that the content analysis application should be able to detect. According to the Nyquist sampling theorem [Pratt 1991], any function of time (e.g., stream of high-level concepts) whose highest frequency is W can be completely determined by sampling at twice the highest frequency, $2W$. In other words, if a stream of high-level concepts is sampled at a frequency less than twice the frequency of the finest temporal details in the stream, high-level concepts may be missed. Hence, the value of the temporal resolution dimension determines the required sampling frequency of the media streams to be analysed, i.e., the frame rate in video analysis.

The *latency* dimension specifies the maximum acceptable elapsed time from an event occurs in the real world until it is reported by the appropriate classifier algorithm. For real-time video content analysis applications that are interactive or feedback control based, this dimension is important.

In general, different classes of quality of service can also be identified, varying from best effort service to guaranteed service. The latter class requires support from the system in terms of resource reservation and admission control, while the former does not. Although problems of resource reservation and admission control have been studied for a long time, their solution has not generally been integrated into more general-purpose operating systems and networks. We therefore restrict the class of considered processing platforms, to general-purpose ones without special real-time processing capabilities. However, we do assume that we have some level of control over the load of competing applications in the processing environment. Furthermore, we believe our results can easily be adapted to take advantage of processing platforms providing real-time scheduling policies.

2.3 Application Architectural Requirements

It seems evident that the resource requirements for the application domain of real-time video content analysis are very challenging and will most likely remain so in the near future. This calls for an architecture that is scalable in the sense that the performance of the application scales well with the amount of processing resources allocated to it. Scalability is required in order to be able to cope with increasing QoS requirements and coordinated analysis of an increasing number of media streams. A scalable application architecture can generally only be obtained by adopting distribution as its basic principle. Scalability of distributed applications is usually achieved by parallelizing application algorithms and distributing the processing of their parts to different processors. The relative complexity of streaming, filtering/transformation, feature extraction, and classification depends on the application. Therefore, the architecture 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, the feature extraction and the classification may become processing bottlenecks. A scalable interaction mechanism which supports such independent parallelization is also required.

The level of accuracy that can be supported by a video content analysis application depends on the misclassification behavior (error rate) of the selected con-

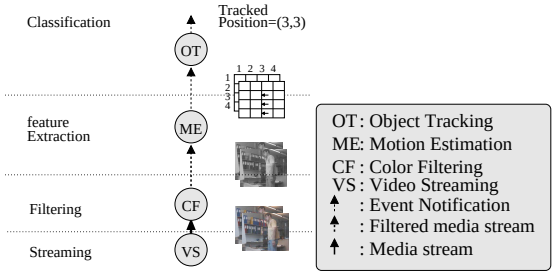

Fig. 2. The functional decomposition of the real-time object tracking application.

figuration of feature extractor and classifier algorithms. Hence, configurations of such algorithms must be carefully selected based on the desired level of accuracy. However, selecting configurations of algorithms which give a high accuracy might result in increased processing time since configurations of algorithms with better accuracy usually require more processing cycles than configurations with poorer accuracy. Therefore, algorithms that can be used to decide whether a QoS requirement can be satisfied in a given distributed physical processing environment are needed. This will include search for an appropriate configuration of feature extractor and classifier algorithms that provides the desired accuracy and that can be allocated to different processors in such a way that the requirements for latency and temporal resolution are fulfilled.

Reduced latency and improved temporal resolution may be achieved in a scalable distributed architecture by allocating independent tasks to different processors. A further improvement in temporal resolution may be achieved by deploying *dependent tasks* as pipelines also on different processors. I.e., in a maximally distributed video processing pipeline, a frame rate of R may be sustained if no task in the pipeline has an average processing time per frame exceeding $1/R$.

In the next subsection we describe some example task graphs. These will be used to exemplify the functionality of the different parts of our application architecture.

2.4 Example Task Graphs

A composition specification of a content analysis application is a decomposition of the video content analysis into tasks. The tasks form a task graph, as discussed above. The granularity of the decomposition should be a modifiable parameter, because the appropriate granularity depends on the processing environment at hand and in particular on the number of processors available.

Figure 2 illustrates the functional decomposition 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 an algorithm doing video stream decoding and color-to-grey level filtering. The filtered video frame is divided into $m \times n$ blocks (video segments) before a motion estimator calculates their motion vectors. The block motion vectors are then received by a classification task (a so-called particle filter) and used for object detection and tracking.

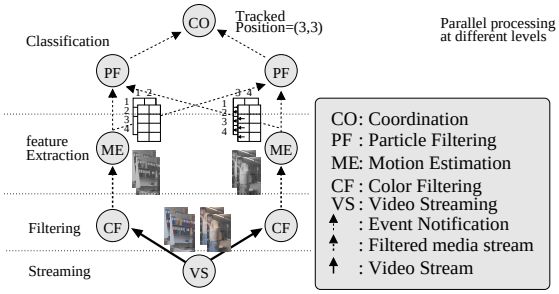

Fig. 3. A configuration of the real-time object tracking application where the computation at several levels is parallelized.

We base our example application on motion vector calculation and particle filtering, because these techniques are recent and promising approaches to object/region tracking in video. To elaborate, calculation of motion vectors (also called optical flow) is a typical pre-processing step in tracking of moving objects/regions[Okada et al. 1996; Bors and Pitas 2000], and the particle filter is a promising approach to object-tracking which allows, e.g., simultaneous tracking and verification [Li and Chellappa 2002].

The content analysis task graph in Figure 2 can be executed as a pipeline, where each level of the chain is executed in parallel. For instance, the application can be executed on four processors, where the streaming is conducted from one processor, the filtering is executed on a second processor, the motion estimation is conducted on a third processor, and the classification on a forth processor. Such distribution allows an application to take advantage of a number of processors equal to the depth of the hierarchy.

Figure 3 illustrates a task graph with a finer granularity. The finer granularity has been achieved by independently decomposing the filtering, feature extraction, and classification into pairs of two tasks. Such decomposition opens up for focusing the processing resources on the processing bottlenecks at hand. For instance, the motion estimation could be conducted on two processors, while the classification, i.e. particle filtering and coordination, can be conducted on three processors.

3. APPLICATION COMPOSITION

In this section we present the algorithms and mechanisms supporting QoS-aware application composition. Input to the composition process is a task graph model of the video content analysis application and meta data describing both the resource requirements of each task and the processing environment. For a given input, the process consists of the following two steps:

(1) Fine grained trading of accuracy against latency/temporal resolution. The starting point is a "brute force" task graph. By a "brute force" task graph we shall mean a task graph that contains the streaming, filtering, feature extraction, and classification tasks deemed relevant, without regard of the processing

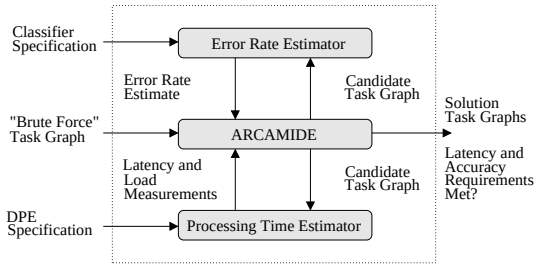

Fig. 4. Structure of algorithm for controlling the accuracy and latency dimensions.

resources available. Tasks are removed iteratively from the task graph until either the latency/temporal resolution requirement is met (success) or the accuracy falls below the required level (failure).

(2) Scalable deployment and execution of the resulting task graph on multiple processors in a distributed processing environment.

In the following, we first present the ARCAMIDE algorithm used for composition, i.e., step one above. Then we discuss support for step two, i.e., scalable solutions for communication, filtering/transformation, feature extraction, and classification.

3.1 Determining Accuracy and Latency

ARCAMIDE is a technique for fine grained trading of accuracy against latency/-temporal resolution. As shown in Figure 4, ARCAMIDE includes the following functionality:

—An *error rate estimator* measures the content analysis error rate of candidate task graphs based on a specification of the classification goal (e.g., object tracking).

—A *processing time estimator* measures the latency and parallelizability of candidate task graphs, given a specification of the distributed processing environment at hand.

—The *ARCAMIDE algorithm* systematically removes tasks from the task graph, trying to reduce latency while at the same time minimizing loss of accuracy.

Based on the above functionality, ARCAMIDE seeks to produce a sequence of solution task graphs that provide a wide range of application composition latency/error rate tradeoffs. Let us now turn to the details of ARCAMIDE by considering how the sequence of solution task graphs actually is produced.

3.1.1 *Task Graph- and Distributed Processing Environment Specification.* As shown in Figure 4, ARCAMIDE takes a "brute force" task graph as input. The brute force task graph contains the S, F, E, and C-tasks deemed relevant for the application. We assume that each task in the graph is annotated with its processing time (the time elapsed from a task receives input until it has produced its output). Furthermore, each edge is assumed annotated with the size (in bytes) of the data communicated along the edge.

ARCAMIDE also needs a specification of the processing environment at hand, i.e., the number and class of processors available as well as the network latency and bandwidth between each pair of processors. To simplify the estimation of latency, we assume that communication is contention free and that network latency and bandwidth do not change. These assumptions are introduced to avoid the additional complexity caused by communication contention, routing, etc. (which are not the focus of this paper), while they still allow handling of a significant class of distributed processing environments (e.g., dedicated computers connected in a dedicated switched LAN).

3.1.2 *Dynamic Bayesian Networks and Error Rate Estimation.* In order to reason about accuracy, ARCAMIDE also needs a specification of the classification goal of the application. We specify classification goals by means of dynamic Bayesian networks (DBNs) [Jensen 2001]. DBNs represent a particularly flexible class of pattern classifiers that allows statistical inference and learning to be combined with domain knowledge. Indeed, DBNs have been successfully applied to a wide range of video content analysis problems [Chang and Sundaram 2000; Naphade and Huang 2002; Garg et al. 2000].

The successful application of DBNs can be explained by their firm foundation in probability theory, combined with the effective techniques for inference and learning that have been developed. For instance, DBNs can learn to automatically associate high-level concepts to video segments from manually annotated video segments, i.e., a training set of concept/feature associations. Generally stated, training consists of finding a more or less accurate mapping between feature space and high-level concept space, within a hypothesis space of possible mappings.

We use a separate set of concept/feature association examples (test set) to estimate the error rate of candidate task graphs. By the *estimated error rate* of a candidate task graph, we shall mean the number of times the DBN misclassifies high-level concepts in the test set, divided by the total number of high-level concepts found in the set. This estimated error rate can be seen as a measure on how accurately the candidate task graph indexes novel video streams.

One important reason for us to base classification on DBNs is the fact that DBNs can classify even when features are missing. In contrast, other types of classifiers, like neural networks and decision trees, must be retrained whenever the feature set changes. Accordingly, by using DBNs exploring the space of possible task graphs becomes more efficient.

3.1.3 *The Processing Time Estimator.* In this section we describe a scheduling algorithm that targets the class of task graphs and distributed processing environments defined in Section 3.1.1. The algorithm is based on generic task graph scheduling principles, as described in [Kwok and Ahmad 1999]. We adapt these generic principles to suit the needs of the ARCAMIDE algorithm. That is, we propose measures of latency and parallelizability that are used to guide task graph pruning.

Our scheduling algorithm seeks to find the allocation of tasks to processors in the distributed processing environment that minimizes latency. As a result of allocating and sequencing tasks to processors, we are able to estimate latency and

temporal resolution. Furthermore, we measure how well the distributed processing environment is taken advantage of, i.e., the parallelizability of the task graph. The respective procedures can be summarized as follows.

We examine the scheduling algorithm first. Let *entry tasks* be tasks without parents and let *exit tasks* be tasks without children in the task graph. We define the *b-level* of a task to be the length of the longest path from the task to an exit node. Likewise, the *s-level* of a task is the length of the longest path from the task to an entry node. The *length of a path* is simply the sum of the task processing times (as given by the task graph) on the path. If multiple classes of processors are available, the average processing time of a task over the available classes of processors is used. Note that when calculating the b-level or the s-level of a task the task itself is included in the path.

A task is either *allocated* to a processor or *not allocated* to a processor. Initially, none of the tasks are allocated to processors. At each iteration of the scheduling algorithm, the non-allocated task with the largest b-level is allocated to a processor. This means that execution of long task graph paths are prioritized before execution of short task graph paths. The main reason behind this strategy is that the longest task graph paths often determine the latency of the task graphs when multiple processors are available, and accordingly should be executed as early as possible.

When a task is to be allocated to a processor the task is scheduled at the earliest *start time* possible. The task may be started when the processor becomes available after previous processing, and the task receives the data produced by its task graph parents. The scheduled *stop time* of a task is simply the sum of its scheduled start time and its processing time (specified by the task graph). A task receives data from a task graph parent at the scheduled stop time of the parent if the two tasks are located on the same processor. Otherwise, the communication time of the data must be added to the data receive time.

When allocating the non-allocated task with the largest b-level to a processor, the processor allowing the earliest task start time is selected. This corresponds to a greedy step towards the goal of minimizing the estimated latency of the task graph. Hence, the processor selection is determined by the location of the tasks parents as well as the communication time of the data communicated from the tasks parents to the task itself.

The above procedure is repeated until all the tasks have been allocated to processors. Based on the resulting scheduling of tasks, we estimate latency and measure parallelizability as follows. First of all, note that the scheduled stop time of the last task to be executed on a processor is the time the processor becomes ready for further processing. We define the estimated latency of a task graph to be the largest processor ready time. By taking the standard deviation of the processor ready times, we measure how well the scheduling algorithm was able to balance the processing load on the available processors. Accordingly, we use the latter quantity to measure the parallelizability of a task graph.

3.1.4 *The ARCAMIDE Algorithm.* In this section we describe a heuristic search algorithm, the ARCAMIDE algorithm, which prunes a "brute force" task graph in order to offer trade-offs between estimated error rate and latency. Note that the ARCAMIDE algorithm does not remove classification tasks from a task graph.

Without classifiers a task graph will only output low-level features and no high-level concepts. This means that the classifier tasks should be treated as a special case. Therefore, we partition the tasks (in the brute force task graph) into streaming, filtering and feature extraction tasks, hereafter denoted SFE-tasks, and classification tasks, hereafter denoted C-tasks.

If the search for a task graph that does not violate the QoS requirements is to be computationally practical, only a very small number of the possible candidate task graphs may be evaluated. E.g., if there are no edges in a task graph containing n SFE-tasks, there are 2^n possible candidate subgraphs. Consequently, the choice of which subgraphs to evaluate is of paramount importance.

In contrast to evaluating all the possible subgraphs of the "brute force" task graph, the ARCAMIDE algorithm consists of two task selection stages. In both stages of the algorithm, the most inefficient parts of the task graph (when considering estimated error rate and latency) are pruned. Roughly stated, our procedure corresponds to a standard sequential backward feature subset search (see [Dash and Liu 1997]), extended to handle task graphs. The task graph search is performed backwards, rather than forwards, in order to avoid a computationally expensive n-step look-ahead search, made necessary by the task graph data dependencies, and in some cases by complexly interacting features. Obviously, other feature subset search procedures can be applied by the ARCAMIDE algorithm (such as beam search [Mitchell 1997], genetic algorithms, or branch and bound search [Dash and Liu 1997]) by extending them to handle task graphs. However, due to its simplicity, computational efficiency, and goal-directed behavior, we here apply a sequential backward task graph search procedure.

The ARCAMIDE algorithm takes as input a set of SFE-tasks, a set of C-tasks, a task irrelevance threshold i, and a parallelizability threshold c. The irrelevance threshold is used for removing tasks irrelevant to the content analysis goal in order to reduce the task graph search space. To elaborate, the error rate achieved by each SFE-task on its own (including its task graph descendants) is estimated. If the resulting estimated error rate is not significantly better than what is achieved with pure guessing, i, the SFE-task is removed along with its descendants. This completes stage one of the ARCAMIDE algorithm.

Let T be the tasks of the current task graph (after previous pruning). In stage two of the ARCAMIDE algorithm, tasks are pruned iteratively from T. Ideally, the least efficient task should be pruned from the task graph at each iteration. The idea is to remove the most costly and/or inaccurate tasks so that the remaining tasks can be completed quicker (improves latency) at minimal loss of accuracy. However, in some cases, removing the least efficient task does not improve latency at all. This occurs when critical paths in the task graph hinder the scheduling algorithm taking advantage of the available processors. Then, sometimes even highly accurate tasks must be pruned before latency can be further reduced.

We resolve the above problem by basing pruning on the parallelization threshold c. Essentially, the threshold controls whether critical paths are to be shortened or the least efficient tasks are to be removed from iteration to iteration. If the standard deviation of the processor ready times (see Section 3.1.3) currently is larger than c, we take this fact to mean that the scheduling algorithm is not able

to take advantage of the available processors due to the length of critical paths. Accordingly, only SFE-tasks at the end of these paths are considered for removal. If, on the other hand, the standard deviation is less than or equal to c, we take this to indicate that the scheduling algorithm is able to balance the processing load between the available processors. Then, the least efficient task is removed from the task graph along with its descendants.

Among the tasks considered for removal (either all or those at the end of critical paths), the task t that minimizes the following efficiency function:

$$ef(t, T) \equiv \frac{er(T \setminus (\{t\} \cup desc(t, T))) - er(T)}{la(\{t\} \cup desc(t, T))}$$

is removed from T in addition to its task graph descendants in T, $desc(t, T)$. Here, $er(T)$ denotes the estimated error rate of task set T, and $la(T)$ denotes the estimated latency of task set T. In short, the efficiency function rewards tasks which contribute to maintaining the estimated error rate of T and punishes tasks that are computationally expensive.

When the ARCAMIDE algorithm stops, it has generated a sequence of task graphs with different estimated error rate/latency trade-offs. These task graphs are sought configured to fully utilize the specified processing environment. The task graph which best fits the provided QoS requirements (in terms of latency, temporal resolution, and accuracy) is selected for deployment in the actual processing environment, as discussed in the following subsection.

3.2 Scalability

Distributed and parallel processing of the task graph suggested by the ARCAMIDE algorithm allows QoS requirements to be satisfied by making available more processing resources. The unit of deployment in our architecture is components, which may encapsulate one or more tasks. Application efficiency depends on scalable solutions for communication, video streaming, filtering/transformation, feature extraction, and classification, each described in the following.

3.2.1 *Event-based Component Interaction.* From Figure 1, 2, and 3, it should be clear that components interact in different ways, such as one-to-one, one-to-many (sharing or partitioning of data), many-to-one (aggregation), and many-to-many. Event-based interaction is well suited in this respect and is additionally characterized by lack of explicit addressing, indirect communication, and hence loose coupling[Eugster et al. 2003]. Clients connect to the event notification service and publish information in so-called event notifications or express their interest by subscribing. Event-based systems differ with respect to the data model for the notifications and the expressiveness of the subscription language. Content-based systems[Segall et al. 2000; Opyrchal et al. 2000; Pietzuch and Bacon 2002; Carzaniga et al. 2001] offer most expressiveness and hence flexibility. Typically, each notification contains a set of attribute/value pairs and clients subscribe to notifications of interest by specifying predicates over these attributes and values.

In our architecture, components connect to and communicate through an event notification service, as illustrated in Figure 5. Consequently, a component does not need to know if notifications have been generated by a single or a number of com-

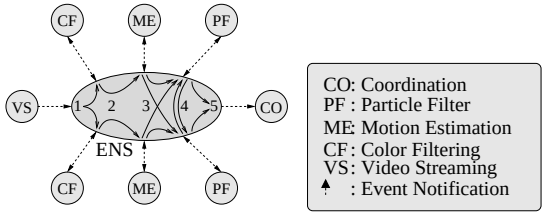

Fig. 5. Inter component communication for the application configuration in Figure 3. The components connect to and interact through a content-based event notification service, labeled ENS.

ponents, nor the location or the identity of the other components. The bindings between components are loose and based on what is produced rather than by whom. The what- rather than whom-characteristics of event-based communication is a key factor for achieving flexible parallelization and distribution. As an example, assume that each PF component subscribes to notifications covered by the following subscription: (src=vs func=me). Assume further that each ME component publishes the calculated motion vectors as notifications, e.g., (src=vs func=me time=[t,dt] block=[x,y] vector=[dx,dy]). The event notification service is then responsible for forwarding these notifications to the PF components, as indicated by label 3 in Figure 5. As a result, from a ME component's point of view it does not matter if there is a single or a number of components interested in the published notifications. Similarly, from a PF component's point of view it does not matter if the motion vectors have been calculated by a single or a number of ME components. This illustrates that event-based interaction enables independent parallelization of the different levels in the content analysis hierarchy.

Scalable content-based publish/subscribe systems are realized as overlay networks of content-based routing nodes, i.e., servers. Each server manages content-based routing tables and forwards notifications towards clients having matching selection predicates. The distributed content-based event notification service used in our project is based on Siena[Carzaniga et al. 2001]. We have extended Siena with IP multicast support in order to support applications requiring high notification rates in combination with a large number of interested clients within a smaller region, such as a LAN or an administrative domain. In short, a mapping from the "event notification space" to IP multicast addresses is specified. When a client publishes a notification through a server, the server maps the notification to an IP multicast address. The notification is thereby efficiently forwarded to all servers which host clients which have matching subscriptions. Note that the inter server communication is transparent to clients. The mapping specification may also be changed on the fly to better fit the current flow of notifications. With our extension of Siena, a client may publish several thousand notifications, carrying several MBytes of data per second, more than sufficient for streaming high quality video.

3.2.2 *Video Streaming.* Real-time video is quite challenging with respect to processing requirements, the massive amounts of data, and the imposed real-time requirements. A video streaming source which must handle each receiver individually will not scale. A heterogeneity challenge also arises when each receiver would like

Fig. 6. Screenshots of three video receivers with different subscriptions, illustrating the effect of selecting: (left) only luminance, lowest quality and only some regions, (middle) only luminance, only chrominance, and full quality color for different rows, and (right) only luminance and low quality, except for a region having full quality and colors.

to receive only a part of the video signal, due to limitations in capacity or interest. The motivation for parallel processing is the distribution of workload by partitioning the video data spatially (e.g., regions within a frame) and/or temporally (e.g., every n'th frame). By receiving only what is needed, both network and processing resource consumption can be reduced and efficiency maintained.

Our approach to this challenge is to exploit content-based publish/subscribe systems for fine granularity multi-receiver video streaming[2]. Each video frame is encoded and published in a number of notifications, each encapsulating only a small part of the full video signal. Each such notification is routed independently by the content-based event notification service to all clients expressing interest in this particular part. Hence, each video receiver may independently customize the video signal along different video quality dimensions, such as region of interest, signal to noise ratio, colors, and temporal resolution. Consequently, efficient delivery is maintained, in terms of network utilization and end node decoding requirements. Figure 6 illustrates that different video receivers may independently subscribe to and thereby select different parts of the video signal. (If the images are unclear in the printed copy, please refer to the electronic version.) A video receiver may even specify different temporal resolutions for different regions within a frame. This flexibility reduces the need for filtering by the application to some degree, e.g., separating the luminance and the chrominance part of the video signal by filtering is not necessary, because this is already handled by the video streaming scheme.

3.2.3 *Filtering and Transformation.* If the fine grained selectivity along the different video quality dimensions is not able to provide an E component with what it needs directly, filtering and transformation is necessary. In other words, filtering and transformation bridge the gap between what a S component offers and an E component can handle. Scalable filtering and transformation requires that a F component may process only some part of a video stream, spatially and/or temporally. In our approach a F component may, e.g., subscribe to only some regions of the video stream, as illustrated in Figure 5, labeled 1. The filtered and transformed

[2]An open source prototype is available from the project web pages[DMJ 1999].

regions are then published as notifications, labeled 2 in the same figure. As a result, filtering and transformation is efficiently distributed and parallelized.

3.2.4 *Feature Extraction.* A feature extraction algorithm operates on video segments from the filtering and transformation level, e.g., video frame blocks, and extracts quantitative information, such as motion vectors and color histograms.

A scalable solution for feature extraction requires that E components may process only some part of a video stream. 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. Additionally, in general the algorithms may be arbitrarily complex.

Feature extraction algorithms for video often operate locally on image regions, e.g., calculating motion vectors, color histograms, and texture roughness. In our architecture, spatial parallelization and distribution of such feature extractors are supported by a block-based approach. Our implementation of a motion estimation component calculates motion vectors for only some of the blocks in a video frame. The motion vectors are published as notifications and forwarded by the event notification service to all interested subscribers, illustrated by label 3 in Figure 5.

3.2.5 *Classification.* The final logical level of our architecture is the classification level. At the classification level each video 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.

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 processor, but accurately tracking the position of multiple people as well as their interactions (talking, shaking hands, etc.) could require several processors. Multiple video streams may increase the content analysis complexity even further. In short, when the classifier is running on a single processor, the classification may become the processing bottleneck of the content analysis application.

The particle filter (PF) [Liu and Chen 1998] is an approximate inference technique that allows real-time DBN-based video content analysis. In the following we briefly describe our use of the PF in more detail. Then we propose a distributed version of the PF, and argue that the communication and processing properties of the distributed PF allow scalable distributed classification, independent of distribution at the other logical levels.

Our PF is generated from a dynamic Bayesian network specifying the content analysis task. During execution the PF partitions the video stream to be analysed into time slices, where for instance a time slice may correspond to a video frame. The PF maintains a set of particles. A single particle is simply an assignment of a content class to each video 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 suppressed. When a new time slice is to be analysed, each particle is independently extended to cover new video segments, driven by the dynamic Bayesian network specification. In order to maintain a relevant set of particles, unlikely particles are then 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 object tracking. Note that apart from the particle replacement, a particle is processed independently of other particles in the PF procedure.

In order to support scalability, we propose a distributed version of the PF. The particles of the single PF are parted into n groups which are processed on n processors. An event based communication scheme maintains global classification coherence. The communication scheme is illustrated in Figure 5 and discussed below. A coordinator (CO) component and n PF components cooperate to implement the particle filter. Each PF component maintains a local set of particles and executes the PF 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 Figure 5). Then, each PF component executes the PF 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 video segment content classes to the coordinator (label 5 in Figure 5) which estimates the most probable content class of each video segment.

In the above communication scheme only $2n$ messages are submitted per time slice. As shown empirically in [Granmo et al. 2003], by only submitting a single particle per message no loss of accuracy is detected in the object tracking case.

4. EMPIRICAL RESULTS

In this section we first study the effectiveness of the application composition strategy used by the ARCAMIDE algorithm. To render the problem challenging and realistic, we simulate highly noisy features that interact both spatially and temporally. Furthermore, we extend our object tracking example task graph so that different parts of the task graph are suitable for different levels of parallel execution. This means that what tasks should be pruned from the task graph depends on the level of parallelization. In short, we want to evaluate the ARCAMIDE pruning strategy under particularly challenging conditions. We then deploy and execute a selection of object tracking task graphs. The purpose is to examine whether our architecture allows the latency and temporal resolution of an application to be improved by increasing the number of processors available to the application. Furthermore, we want to demonstrate that timeliness can be improved by reducing the number of features extracted from each video frame, at the potential cost of reduced accuracy. In other words, we aim to examine the main premises of the ARCAMIDE algorithm, not to evaluate the accuracy of the tracking method used.

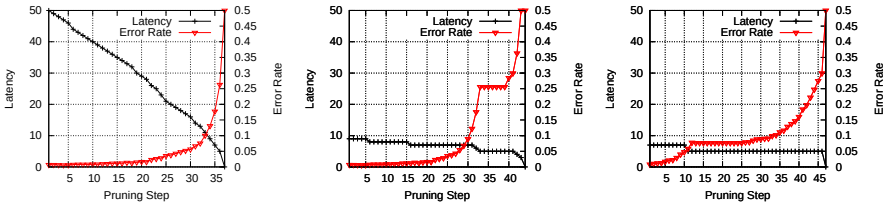

Fig. 7. The estimated latency and error rate (y-axis) after each task removal (x-axis) for different configurations: 1 processor (left), 10 processors (middle), and 100 processors (right).

4.1 Application Composition by means of Task Graph Pruning

We evaluation the ARCAMIDE algorithm empirically based on simulating features extracted from 8×8 video frame blocks. The simulation is constructed from the object tracking example application shown in Figure 3, however, we refine the content analysis problem to recognition of whether an object passes from the left to the right or vice versa. Furthermore, we also add a color histogram calculation task and a texture calculation task to each video frame block.

Thus, we have five different types of SFE-tasks related to each video frame block: streaming, color-to-grey level filtering, motion estimation, texture calculation, and color histogram calculation. These have different content analysis and processing characteristics. In our simulation, when using motion to detect an object in a video frame block, the probability of false positives and false negatives are assumed to be 0.3. Likewise, when using texture the probability of false positives is assumed to be 0.5 and the probability of false negatives is assumed to be 0.3. When using color the latter probabilities are reversed. Color-to-grey-level filtering only produces intermediate results used in the motion estimation. Obviously, the specified probabilities depend on the environment (e.g., sunshine and darkness) and are here set to reflect rather difficult environment conditions. Finally, the processing time of the streaming task is set to 3 ms, and the processing time of the other tasks are set to 1 ms. The transmission time of a video frame block (either color-to-grey level filtered or not) across the network is considered to be 1 ms.

To evaluate the ARCAMIDE algorithm we trained two 10-state Hidden Markov Models (one for each object passing direction) on 1000 simulated video frame sequences of objects moving from the left to the right and 1000 sequences of objects moving from the right to the left. Here, an object appears on average twice in each video frame block of the two center rows when passing the camera view. We used another independent set of simulated video frame sequences (of identical size) to prune the task graph.

When trading off estimated error rate and latency, the ARCAMIDE algorithm behaves as shown in Figure 7 for configurations targeting 1 processor (left), 10 processors (middle), and 100 processors (right). When selecting SFE-tasks for 1 processor, we see from the left plot that the estimated latency initially can be reduced with little increase in estimated error rate (while inaccurate SFE-tasks are removed). However, when e.g. SFE-tasks near the first and eighth video frame block columns (the object entry regions) must be removed, the estimated content

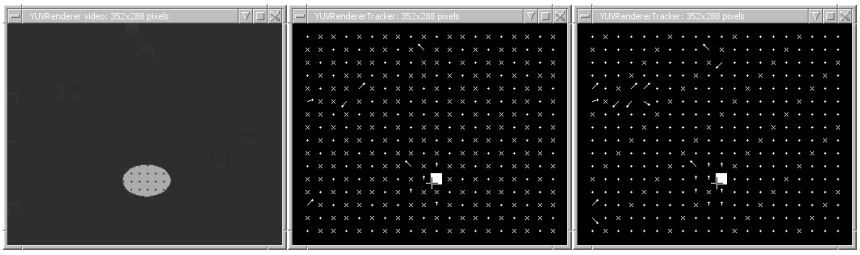

Fig. 8. Left: Video input as seen by the motion estimation tasks. Middle: Output from the motion estimation tasks when extracting 40 % of the features. Right: Output from the motion estimation tasks when extracting 80 % of the features. In the screenshot images, x indicates a skipped block, · illustrates no motion, arrows represent the motion vectors, + represents the object's true position, while the tracked position is drawn as a white square.

analysis error rate increases more dramatically. When introducing 10 processors, we see from the middle plot that the estimated latency is reduced in steps — the processing time of all the processors must be reduced before the latency can be reduced. Also note that the data dependencies between color-to-grey level filtering and motion estimation make these tasks the target of removal from removal number 28 and thereafter. When considering 100 processors, as seen in the right plot, initially only the removal of motion SFE-tasks will reduce the processing time due to the dependency of motion SFE-tasks on color-to-grey level filtering tasks. In the right plot there are mainly two interesting task graphs; one containing all the relevant SFE-tasks and one containing only texture and color SFE-tasks.

4.2 Task Graph Case Studies with Timeliness, Accuracy, and Scalability Results

Performance measurements for the experiments with a distributed object tracking application are presented in this section. Our aim was to investigate how accuracy, temporal resolution, and latency are affected by deploying different task graph configurations on a varying number of processing nodes.

In all experiments, a single component streamed the video data in CIF resolution (352×288 pixels). In order to measure accuracy precisely, full control of the video content, e.g., object position and movement, was needed. Hence, synthetic video data was used. The video background was static, but some noise was added to each video frame. The real object position and a timestamp was published each time a new video frame was captured from the synthetic video source, but before video compression. A logging component subscribed to this real object position as well as the tracked position from the coordinator. Precise timing of the end-to-end latency required a single reference clock. The logging component therefore executed on the same computer as the video streaming component. The log data were written to file for off-line analysis. Regarding accuracy, we defined the tracked position as correct if the position was within the true position block or in any other neighboring blocks (16×16 pixels block size). Otherwise we defined it as a misclassification.

The motion estimation task was handled by one or more motion estimation components. Each component was configured to process a region within each frame,

but subscribed to a slightly larger region in order to calculate motion vectors for the border blocks. Consequently, some parts of the frame is decoded by several components during parallelization. The components subscribed to only the luminance part of the video signal, eliminating transport and decoding of color information. An example video image is shown in Figure 8, left. The search area was ±6 pixels, both horizontally and vertically, in other words a search area of 169 displacements. A full search was always performed, i.e., enforcing worst case behavior. The number of blocks processed in each frame was 20×16, i.e., border blocks were not processed. Note that a full frame needs to be buffered before motion vectors can be calculated based on the previous and the current frame, introducing a frame rate dependent delay.

The task graphs included one or more particle filtering components and a single coordinator component. 2000 particles were evenly distributed among the particle filters. Each particle filtering component subscribed to both motion vectors and particles submitted by the other particle filters. The particle filters published object position probability distributions, which the coordinator subscribed to.

The distributed content-based publish/subscribe system handled all communication, including the streamed video data. The mapping specification used, mapped the notification space to 18 different IP multicast addresses. Each video frame was divided in 11 columns, each mapped to a separate IP multicast address.

The distributed computing platform was realized by standard 1667MHz dual AMD Athlon PCs running Debian GNU/Linux. The PCs were connected by 100Mbps switched Ethernet. Java Standard Edition build 1.4.2_03-b02 was used for compilation and execution.

4.2.1 *Improving Timeliness by Parallel Processing.* In the first experiment, we increased the frame rate and hence the temporal resolution by deploying the application on an increasing number of processing nodes. The accuracy and end-to-end latency were measured for the different configurations. The CPU consumption for each component type was also measured and reveals the scalability of the application in particular, but also indicate the scalability of the architecture in general. Note that the number of CPUs reported does not include the separate computer hosting both the video streaming and the logging component.

The first configuration was executed on two processors; the motion estimation component was executed on one processor and the particle filter and coordination component were executed on another processor. In order to take advantage of additional processors, new configurations were created by stepwise adding one motion estimation component and one particle filter component, each executed by a dedicated processor. The configuration illustrated in Figure 3 was executed on 4 processors. Similarly, in the 10 processor configuration five motion estimation components and five particle filtering components were deployed.

From Table I it can be seen that the accuracy remains more or less constant as the frame rate increases. This is as expected, because the application performs the same work, although at a faster rate and in a distributed fashion. Additionally, the end-to-end latency decreases as motion estimation and particle filtering is processed in parallel. The shorter inter frame delay also contributes to the decrease in latency For the 10 CPU case, there is an increase in median latency. Our log traces seem

Table I. The frame rate, accuracy, latency, and CPU load for different configurations of the real-time object tracking application.

	Number of processors			
	2	4	8	10
Overall Frame Rate	5	10	20	25
Accuracy (%)	98.1	99.5	99.2	98.2
End-to-End Latency (ms)				
Min	475	244	133	117
5th percentile	488	255	144	137
Median	499	266	179	232
95th percentile	596	337	341	589
Max	829	535	590	803
CPU Load (%)				
Streaming	12	28	60	70
Feature Extraction	69	69	73	75
Particle Filtering	69	71	76	78
Coordination	2	2	3	5

to indicate that parts of the system are approaching the saturation point, causing queues which retain data due to temporary overloads.

The CPU requirements for each motion estimation component increases slightly as the frame rate increases, and is most likely due to the video decoding overhead inherent in motion estimation. However, this is a significant improvement compared to results presented in [Mayer-Patel and Rowe 1999a] and [Eide et al. 2003], where each component decodes a full video stream sent over IP multicast before processing only a region within each frame (cf. Section 5.2.7). The particle filtering part of the computation shows a similar slight increase in CPU load as processors are added. The video streaming component was executed by a single processor, which explains the increase in CPU consumption. Due to its low complexity, the coordination task consumes small amounts of processing resources

4.2.2 *Improving Timeliness by Task Graph Pruning.* In the second experiment, different task graph configurations were generated. The configurations differed with respect to the percentage of features calculated for each video frame. As an example, in the 20% configuration only 64 motion vectors were calculated for each frame, out of 320. Figure 8 shows two example frames, where respectively 40% and 80% of the features have been extracted. Three dual CPU computers were used in this experiment. One computer hosted the video streaming and logging components. Another computer hosted two motion estimation components, while the third computer hosted two particle filter components and the coordinator component. E.g., in the 20% configuration, each motion estimation component received the video data required for computing 32 motion vectors.

Table II shows the accuracy, latency, and CPU load for the different task graphs. Accuracy increases as more features are extracted, and accordingly, more information is given to the particle filters about the object position. Latency and CPU load decrease when less features are extracted. Essentially, the amount of work each processor has to do is reduced, because tasks are being removed. Furthermore, the particle filters perform less calculations when processing fewer features, which re-

Table II. Accuracy, latency, and CPU load are all influenced by the percentage of features extracted from each video frame.

	Percentage of features extracted					
	0	20	40	60	80	100
Overall Frame Rate	10	10	10	10	10	10
Accuracy (%)	0.9	65.3	95.2	95.9	96.8	99.5
End-to-End Latency (ms)						
Min	79	144	166	195	224	244
5th percentile	131	157	181	207	232	255
Median	136	161	186	215	243	266
95th percentile	192	219	243	278	315	337
Max	344	340	357	416	456	535
CPU Load (%)						
Feature Extraction	9	20	33	48	60	69
Particle Filtering	6	23	33	47	61	71

duces the load of particle filtering. Coordination and streaming are unaffected by the percentage of features extracted and are therefore not included in the table.

To conclude, we have demonstrated that our architecture allows the latency and temporal resolution of an application to be significantly improved by utilizing an increasing number processors. Furthermore, we have demonstrated that timeliness can be improved by reducing the number of features extracted from each video frame at the cost of reduced accuracy, as assumed by the ARCAMIDE algorithm.

5. COMPARISON WITH RELATED WORK

This section provides a discussion of related work and is structured as three subsections: one-to-many communication, media streaming and processing, and classification.

5.1 One-to-Many Communication

Efficient one-to-many communication has been studied extensively over several decades. Here we consider technologies suited for the targeted application domain, where real-time, and hence efficient dissemination, is important.

5.1.1 *Multicast and Group Communication.* Lack of group communication support in the IP communication suite inspired a number of research efforts in the 1980s and lead to technologies for reliable group communication[Birman et al. 2000] and IP multicast[Deering and Cheriton 1990]. Reliable group communication allows, e.g., replication of data and computation for improved robustness. The development of IP multicast was driven by a need to also support real-time streaming of video and audio data. By pushing multicast functionality into the network, link stress could be reduced to a minimum.

Despite tremendous efforts, IP multicast is still not an ubiquitous service. In the past few years, peer-to-peer systems[Crowcroft and Pratt 2002] have emerged as a viable approach for implementing multicast functionality (e.g., group membership control, routing, and packet forwarding) solely in the end systems[Chu et al. 2002]. Such overlay networks may also be customized for application specific requirements, e.g., trading off reliability for increased bandwidth and reduced delay. However, the

main challenge is to keep the link stress down[Castro et al. 2003]. These group-based systems provide a level of indirection between senders and receivers.

5.1.2 *Event-Based Communication.* Compared to group-based systems, event-based systems[Eugster et al. 2003] provide finer granularity for expressing interest. Most flexibility and expressiveness are offered by content-based event-notification services[Segall et al. 2000; Opyrchal et al. 2000; Pietzuch and Bacon 2002; Carzaniga et al. 2001]. The major difference compared to group-based systems is that data is forwarded based on the *content*, not on explicit group addresses. Similarly to group-based systems, the construction and maintenance of an efficient overlay network for content-based routing is challenging. Recently, peer-to-peer technologies have also been exploited in this respect[Pietzuch and Bacon 2003; Terpstra et al. 2003].

5.1.3 *Discussion.* Group-based systems, such as IP multicast, forward data based on group membership. This is a rather course grained way of expressing interest, but the situation can be improved somewhat by using a number of groups. However, a major drawback is the difficulty of determining mappings between content and groups. Additionally, the mappings become relatively static.

The finer granularity and the improved flexibility, by having an additional level of indirection, motivated us to exploit content-based publish/subscribe systems for real-time video content analysis. To the best of our knowledge, existing content-based publish/subscribe systems do not take advantage of native multicast support. Therefore, we extended Siena[Carzaniga et al. 2001] with IP multicast support[Eide et al. 2003] in order to provide efficient dissemination of notifications. Other group-based systems may have been used instead of IP multicast, e.g., for improving robustness. Altogether, the added complexity, compared to using group-based communication directly, seems manageable. The experimental results presented in this paper demonstrate the benefits and potential of our approach.

5.2 Media Streaming and Processing

5.2.1 *Commercial Frameworks.* Several commercial frameworks for developing multimedia applications exist, including the Java Media Framework (JMF) [Sun Microsystems 1999] and DirectShow[Blome and Wasson 2002] from Microsoft. Since the functionality is similar, only JMF is discussed. Applications are modeled as a flow graphs in JMF — nodes represent media handling modules, while the edges represent media flows. JMF performs low level media tasks, such as capture, transport, streaming, (de)multiplexing, (de)coding, and rendering. Access to the raw media data is provided by means of a plug-in architecture which allows integration of customized processing. Standard streaming protocols are also supported. Programmers specify media sources/sinks, using URLs, and JMF internally tries to build a flow graph by connecting different components.

5.2.2 *Infopipe.* The Inforpipe project[Black et al. 2002] aims at simplifying application development within the domain of media streaming. Certain aspects of the communication are made explicit, allowing application level monitoring and adaptation in response to varying resource availability. The defined building blocks, e.g., sources, sinks, buffers, filters, pumps, remote pipes, and split and merge tees, may be composed into Infopipelines. Reasoning about compositions, both with respect

to functionality and QoS characteristics is supported (e.g., validating data flow and determining end-to-end latency). The authors recognize that some QoS properties do not have strict additive behavior, making reasoning very difficult. The ideas from the Infopipe project have influenced the development of GStreamer[Taymans et al. 2004], an open source framework for developing streaming media applications.

5.2.3 *Media Gateway Systems.* In order to bridge the heterogeneity gap created by differences in resource availability, hardware capabilities, and software incompatibilities, media gateway systems have been proposed for streaming[Ooi and van Renesse 2001; Rafaelsen and Eliassen 2002]. These systems are overlay networks — the media gateways are internal network nodes, while senders and receivers are at the edges. Gateways receive media streams from upstream nodes, before forwarding the processed and potentially transformed streams to downstream gateways and receivers. Bandwidth requirements may be reduced by temporal-, spatial-, and signal to noise ratio scaling and the format may be changed. More complex operations may compose streams, e.g., generating picture-in-picture effects. Overlay construction is challenging; the goal may be to minimize (average/maximum) delay, processing, or bandwidth consumption, but receivers may join and leave at any time. In [Ooi and van Renesse 2001] the media processing is represented by small scripts. The computation may be decomposed into sub-computations and then mapped onto gateways, driven by the goal of reducing bandwidth consumption.

5.2.4 *Receiver-driven Layered Multicast (RLM).* An approach for handling both the multi-receiver and the heterogeneity challenge is to combine layered video coding techniques with transport over several multicast channels. A pioneering work in this respect is video coding for receiver-driven layered multicast (RLM) [McCanne et al. 1997]. The video signal is encoded *cumulatively* into a number of layers, each sent on a separate multicast address. The base layer provides the lowest quality, and each additional layer improves the video quality. Clients with low bandwidth connections may register interest in the base layer only, while other clients may subscribe to a number of additional multicast addresses, as bandwidth permits. A layering policy determines the dimension to be refined for each additional layer.

5.2.5 *MediaMesh.* MediaMesh[Amini et al. 2000] is an architecture for integrating isochronous processing algorithms into media servers. Operations on live media streams are supported, e.g., security (watermarking and encryption), time shifting (pause and seek), adaptation, and (de)multiplexing. Media streams are modelled as directed graphs where edges represent communication and nodes represent processing modules. In contrast to the media data, control information flows in both directions - upstream for control requests and downstream for replies. The architecture is QoS aware and resource management support is deterministic. During setup of a flow graph, overall CPU and memory usage is predicted, based on properties associated to each filter module (e.g., CPU and memory requirements). From our understanding, the processing of a flow graph is limited to a single host.

5.2.6 *Parallel Software-only Video Effects Processing System (PSVP).* A research effort for exploiting functional, temporal, and spatial parallelization of video effects is described in [Mayer-Patel and Rowe 1998; 1999a; 1999b]. The target

domain is Internet video. Effects, such as fade, blend, and affine transformations are supported as well as the ability to compose video streams. Effect processing is specified in a high level language and a compiler generates a directed graph, which is then mapped onto the available processors. For spatial parallelization, each processor receives the stream over IP multicast, decodes the full video images, and applies the effects processing on a region. The different frame regions are then combined by a processor into an effect-enhanced video stream. In [Mayer-Patel and Rowe 1999a], performance numbers are presented for a general affine transform effect, executed by one to seven processors. The reported latency for the one processor configuration is 250 ms. Seven processors and spatial parallelization results in 70 ms latency and 50% efficiency, bounded by the overhead due to receiving and decoding the full video stream before applying the effects.

5.2.7 *Discussion.* The graph building processes in both JMF and MediaMesh do not take into account more than one computer. The graph building process in JMF is automatic, implicit, and to the best of our knowledge undocumented. Infopipe supports explicit connections and allows reasoning about distributed compositions. Media gateway systems have internal support for reasoning about some overall goal during construction and maintenance of the media distribution overlay network. However, for reasoning about more high level and non-additive dimensions, such as accuracy, more than the streaming and video processing part of the application must be taken into account. A predictive cost model for reasoning about video effect graphs was planned by the authors of PSVP, but we were unable to find any more details. In contrast, we present both a model and an algorithm for reasoning about the combination of latency, temporal resolution, and accuracy.

In JMF, Infopipe, and MediaMesh components are connected directly, without a level of indirection. This makes (re)configuration and independent parallelization of the different levels in the video analysis hierarchy difficult. Additionally, no support for parallel and distributed processing of video streams is provided. The authors of PSVP acknowledge the difficulty of distributing different parts of a video stream to different processors, due to spatial and temporal dependencies. The authors also recognize that sending a full video stream to all processors gives a significant decoding overhead, confirmed by their experiments. RLM is a significant improvement, but an inherent problem is conflicting user preferences. Since the layering policy is fixed at the sender side, each video receiver is unable to customize the video stream independently. Media gateway systems support stream customization — a gateway may perform any kind of filtering and transformation. E.g., a gateway may partition a video stream spatially as a preparation step for parallel processing. The cost associated with this flexibility is increased network and processing resource consumption and additional delay. Several receivers may also share interest in some part of the video signal, and handling such cases efficiently seems very difficult.

Our experiences with earlier prototypes, based on JMF and streaming over IP multicast[Eide et al. 2003], were a driving factor for us to exploit content-based networking for fine granularity multi-receiver video streaming. In our approach additional processors for partitioning the video data are not needed, since this is handled by the video coding scheme and the content-based publish/subscribe system. Additionally, each video receiver may independently customize the stream,

along different quality dimensions. In effect, CPU and bandwidth consumption is reduced and latency decreased. The performance numbers reported herein show the strength of our approach — the overhead for feature extraction is small and most likely due to the dependencies inherent in motion vector estimation.

5.3 Classification

With respect to classification, we first review application specific- as well as generic approaches that use *task/feature selection* to manipulate classification accuracy and timeliness. We then review representative work that focus on *task scheduling* for controlling timeliness in video content analysis.

5.3.1 *Task Selection.* In the application specific approach found in [Zhou et al. 1998], different scene change detection algorithms are evaluated. The purpose is to identify which algorithm is most appropriate for real-time analysis under different data bandwidths. Based on performance studies, the authors are able to choose the right algorithm for the best performance in each case.

A related approach to resource-awareness is found in [Leszczuk and Papir 2002], where a technique for shot (scene change) detection is accelerated by only processing every n'th pixel in the horizontal and vertical directions. As the number of pixels analyzed is reduced, the scene change detection accuracy will in general deteriorate.

A more generic approach to selecting image content analysis algorithms/tasks is *feature selection*. In [Yang and Honavar 1998] feature selection is based on multi-criteria optimization. Medical diagnosis is pointed out as a motivating example, because not only the accuracy of diagnostic tests (features) are of importance, but also the various associated costs and risks of the tests. The relevance of this work is the introduction of feature/test cost into the feature criterion function.

The technique proposed in [Smits and Annoni 2000] also considers feature extraction cost. A stepwise procedure is suggested, where one feature is added at a time (to an initially empty feature set), until a given target classification accuracy is achieved. When several features achieve the target accuracy, the least costly feature is selected. The aim of the procedure is on-line feature selection. This is in contrast to the off-line approach of [Yang and Honavar 1998] where a potentially more thorough, but also more costly feature subset inspection approach is applied (genetic algorithm based).

5.3.2 *Task Scheduling.* Recent work on task scheduling within video content analysis include [Marcenaro et al. 2001] and [Yang et al. 2003]. [Marcenaro et al. 2001] proposes a physical and a logical architecture for third generation surveillance systems (defined as a distributed multimedia computer system based on multi-sensory input data and which offers automatic scene understanding capabilities). The *physical architecture* is organized as a tree whose nodes are processing entities such as intelligent cameras, hubs and control centers. Links are associated with communication channels. The *logical architecture* is described as the information process necessary to obtain the desired system functionality (i.e., the content analysis application), combined with a hierarchical partitioning of this information process into a set of logical processing tasks — e.g., image acquisition, image change detection, object filtering, and tracking. A main issue is the dynamic allocation of the logical processing tasks to the processing nodes of the physical architecture,

with the purpose of minimizing the overall processing time of each video image. The computational complexity of individual tasks as well as the bandwidth required for data flows between tasks are estimated. These estimates form the basis for evaluating the response time of different candidate task allocations to cameras, hubs, and control centers, so that an optimal allocation can be found.

A similar approach is found in [Yang et al. 2003]. An automatic compile-time scheduler is proposed that schedules the tasks of a video processing application on available processors. The scheduler exploits both spatial (parallelism) and temporal (pipelining) concurrency to make the best use of available machine resources. Two important scheduling problems are addressed. First, given a task graph and a desired throughput (video processing image rate), a schedule is constructed to achieve the desired throughput with as few processors as possible. Second, given a task graph and a finite set of resources, a schedule is constructed such that the throughput is sought maximized.

5.3.3 *Discussion.* The two application specific approaches, i.e., [Zhou et al. 1998; Leszczuk and Papir 2002], allow accuracy to be traded against timeliness, but are rather limited in scope since only scene change detection algorithms are considered. The more generic feature selection approaches [Yang and Honavar 1998; Smits and Annoni 2000] typically apply a too simplified view on task scheduling and parallel processing, compared to the needs in the real-time video content analysis domain. In real-time video content analysis, task execution precedence constraints as well as processing/communication resource restrictions influence how feature selection should proceed, introducing difficulties for traditional feature selection. The task graph scheduling work presented in [Marcenaro et al. 2001; Yang et al. 2003] are similar to our approach. However, these efforts were conducted independently of our work. More importantly, our focus has not been on task scheduling for controlling timeliness alone. Rather, we have focused on a different and previously unaddressed problem, namely, *combining* task graph scheduling with feature selection so that *both* accuracy and timeliness can be controlled.

6. CONCLUSION AND FURTHER WORK

In this paper paper we have presented techniques for QoS-aware application composition and solutions for scalable distributed and parallel processing for the domain of real-time video content analysis.

QoS-aware application composition is the process of determining the deployment configuration of an application to satisfy a given set of QoS requirements. Our solution includes a generic application architecture that allows applications to be independently distributed and parallelized at multiple logical levels of processing, and an application QoS model that can be used to communicate QoS requirements for a particular application instantiation. A salient feature of our QoS-aware composition approach is the combination of probabilistic knowledge-based media content analysis and QoS. For this purpose we proposed an algorithm for balancing the requirements of latency, temporal resolution, and accuracy against the available distributed processing resources.

For classification, a parallel version of an approximate inference technique, known as the particle filter, has been developed. We demonstrated that the parallel particle

filter enables scalable real-time video content analysis based on dynamic Bayesian networks. Independent scalability at multiple logical levels was primarily achieved through the development of a high performance content-based event notification service, the preferred communication mechanism used throughout the architecture. Our experience as reported in this paper, indicates that high-performance content-based publish/subscribe systems are more suitable for real-time distributed and parallel video content analysis, than using group-based communication directly. This paper has also shown how scalable video processing can be achieved by means of a novel fine granularity multi-receiver video coding scheme, which exploits the content-based publish/subscribe system. This new video encoding scheme supports fine granularity selectivity for each video receiver along different video quality dimensions, such as region of interest, signal to noise ratio, colors, and temporal resolution. As a consequence, efficiency can be maintained for parallel processing of video data.

The scalability of our approach was demonstrated through experiments with an implementation of a real-time motion vector based object tracking application. The experiments show that increasing QoS requirements can be met by increasing the number of CPUs used for parallel processing, with only minor overhead.

Overall, we believe this work has shown that it is feasible to provide generic support for developing scalable distributed real-time video content-analysis applications from reusable components, i.e., content analysis tasks. In particular we believe our framework assists the developer in separating functional concerns from non-functional ones thus simplifying the development of this kind of applications.

Further experiments to validate the scalability and extensibility of the architecture can be done along different dimensions, separately or in combination. This includes demonstrating its extensibility with respect to the use of different media types and different feature extractor types concurrently, and the combination of temporal and spatial parallelization.

Future work will also include better support for developing distributed real-time content-based video applications. In particular we are addressing the question of how general purpose software component architectures can be made to support this kind of applications. We will also investigate how complex content analysis applications can take advantage of high-performance computational Grids. Grid systems have not yet been designed to handle real-time applications. In this respect, real-time distributed and parallel multimedia analysis represents a challenging application domain which may influence research in grid technologies.

7. ACKNOWLEDGMENTS

We would like to thank all persons involved in the Distributed Media Journaling project[DMJ 1999] for contributing to the ideas presented in this paper.

REFERENCES

Amini, L., Lepre, J., and Kienzle, M. G. 2000. Mediamesh: An architecture for integrating isochronous processing algorithms into media servers. In *Multimedia Computing and Networking (MMCN'00)*, K. Nahrstedt and W. chi Feng, Eds. Vol. 3969. SPIE, 14–25.

Beymer, D., McLauchlan, P., Coifman, B., and Malik, J. 1997. A Real-time Computer Vision System for Measuring Traffic Parameters. In *Computer Vision and Pattern Recognition (CVPR'97), San Juan, Puerto Rico*. IEEE, 495–501.

Birman, K., Constable, R., Hayden, M., Hickey, J., Kreitz, C., van Renesse, R., Rodeh, O., and Vogels, W. 2000. The Horus and Ensemble Projects: Accomplishments and Limitations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*. IEEE Computer Society Press. 149–160.

Black, A. P., Huang, J., Koster, R., Walpole, J., and Pu, C. 2002. Infopipes: An abstraction for multimedia streaming. *Multimedia Systems 8,* 5, 406–419.

Blome, M. and Wasson, M. 2002. DirectShow: Core Media Technology in Windows XP Empowers You to Create Custom Audio/Video Processing Components. *Microsoft, MSDN Magazine 17,* 7 (July).

Bors, A. and Pitas, I. 2000. Prediction and tracking of moving objects in image sequences. *IEEE Transactions on Image Processing 9*, 1441–1445.

Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. 2001. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems 19,* 3 (August), 332–383.

Castro, M., Jones, M. B., Kermarrec, A.-M., Rowstron, A., Theimer, M., Wang, H., and Wolman, A. 2003. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *INFOCOM*.

Chang, S.-F. and Sundaram, H. 2000. Structural and Semantic Analysis of Video. In *Multimedia and Expo 2000 IEEE*. Vol. 2. 687–690.

Chen, C., Jia, Z., and Varaiya, P. 2001. Causes and Cures of Highway Congestion. *Control Systems Magazine, IEEE 21,* 6 (December), 26–32.

Chu, Y., Rao, S. G., Seshan, S., and Zhang, H. 2002. A case for end system multicast. *IEEE Journal on Selected Areas in Communications (JSAC) 20,* 8 (October), 1456–1471.

Crowcroft, J. and Pratt, I. 2002. Peer to peer: Peering into the future. In *NETWORKING Tutorials*, E. Gregori, G. Anastasi, and S. Basagni, Eds. Lecture Notes in Computer Science, vol. 2497. Springer, 1–19.

Dash, M. and Liu, H. 1997. Feature selection for classification. *Intelligent Data Analysis 1,* 3.

Deering, S. E. and Cheriton, D. R. 1990. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Trans. Comput. Syst. 8,* 2, 85–110.

DMJ. 1999. The Distributed Media Journaling Project. http://www.ifi.uio.no/~dmj/.

Eide, V. S. W., Eliassen, F., Granmo, O.-C., and Lysne, O. 2003. Supporting Timeliness and Accuracy in Real-time Content-based Video Analysis. In *Proceedings of the 11th ACM International Conference on Multimedia, ACM MM'03, Berkeley, California, USA*. 21–32.

Eide, V. S. W., Eliassen, F., Lysne, O., and Granmo, O.-C. 2003. Extending Content-based Publish/Subscribe Systems with Multicast Support. Tech. Rep. 2003-03, Simula Research Laboratory. July.

Eide, V. S. W., Eliassen, F., and Michaelsen, J. A. 2004. Exploiting Content-Based Networking for Video Streaming. In *Proceedings of the 12th ACM International Conference on Multimedia, Technical Demonstration, ACM MM'04, New York, New York, USA*. 164–165.

Eide, V. S. W., Eliassen, F., and Michaelsen, J. A. 2005. Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming. In *Proceedings of the Twelfth Annual Multimedia Computing and Networking (MMCN '05), San Jose, California, USA*, S. Chandra and N. Venkatasubramanian, Eds. Vol. 5680. 155–166.

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. 2003. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR) 35*, 114–131.

Garg, A., Pavlovic, V., Rehg, J., and Huang, T. 2000. Integrated Audio/Visual Speaker Detection using Dynamic Bayesian Networks. In *Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000*. 384–390.

Granmo, O.-C., Eliassen, F., Lysne, O., and Eide, V. S. W. 2003. Techniques for Parallel Execution of the Particle Filter. In *Proceedings of the 13th Scandinavian Conference on Image Analysis (SCIA2003)*. Lecture Notes in Computer Science, vol. 2749. Springer, 983–990.

Gu, X. and Nahrstedt, K. 2005. Distributed Multimedia Service Composition with Statistical QoS Assurances. *To appear in IEEE Transactions on Multimedia*.

Jacobsen, H.-A., Ed. 2003. *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, DEBS 2003, Sunday, June 8th, 2003, San Diego, California, USA (in conjunction with SIGMOD/PODS)*. ACM.

Jensen, F. V. 2001. *Bayesian Networks and Decision Graphs*. Series for Statistics for Engineering and Information Science. Springer Verlag.

Kwok, Y.-K. and Ahmad, I. 1999. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing 59,* 3, 381–422.

Leszczuk, M. and Papir, Z. 2002. Accuracy vs. Speed Trade-Off in Detecting of Shots in Video Content for Abstracting Digital Video Libraries. In *Protocols and Systems for Interactive Distributed Multimedia (IDMS/PROMS 2002)* (November). LNCS, vol. 2515. Springer, 176–189.

Li, B. and Chellappa, R. 2002. A generic approach to simultaneous tracking and verification in video. *IEEE Transactions on Image Processing 11*, 530–544.

Liu, J. S. and Chen, R. 1998. Sequential Monte Carlo methods for Dynamic Systems. *Journal of the American Statistical Association 93,* 443, 1032–1044.

Marcenaro, L., Oberti, F., Foresti, G. L., and Regazzoni, C. S. 2001. Distributed Architectures and Logical-Task Decomposition in Multimedia Surveillance Systems. *Proceedings of the IEEE 89,* 10 (October), 1419–1440.

Mayer-Patel, K. and Rowe, L. A. 1998. Exploiting temporal parallelism for software-only video effects processing. In *Proceedings of the sixth ACM International Conference on Multimedia, September 13-16, 1998, Bristol, United Kingdom*, W. Effelsberg and B. C. Smith, Eds. ACM, 161–169.

Mayer-Patel, K. and Rowe, L. A. 1999a. Exploiting spatial parallelism for software-only video effects processing. In *Multimedia Computing and Networking (MMCN'99), San Jose, California, USA*, D. D. Kandlur, K. Jeffay, and T. Roscoe, Eds. Vol. 3654. SPIE, 252–263.

Mayer-Patel, K. and Rowe, L. A. 1999b. A multicast scheme for parallel software-only video effects processing. In *Proceedings of the seventh ACM International Conference on Multimedia, October 30 - November 05, 1999, Orlando, Florida, USA*, J. Buford, S. Stevens, D. Bulterman, K. Jeffay, and H. Zhang, Eds. Vol. 1. ACM, 409–418.

McCanne, S., Vetterli, M., and Jacobson, V. 1997. Low-complexity video coding for receiver-driven layered multicast. *IEEE Journal of Selected Areas in Communications 15,* 6 (August), 983–1001.

Mitchell, T. M. 1997. *Machine Learning*. Computer Science Series. McGraw-Hill International Editions.

Naphade, M. and Huang, T. 2002. Extracting semantics from audio-visual content: the final frontier in multimedia retrieval. *IEEE Transactions on Neural Networks 13,* 4, 793–810.

Okada, R., Shirai, Y., and Miura, J. 1996. Object tracking based on optical flow and depth. In *Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems*. IEEE, 565–571.

Ooi, W. T. and van Renesse, R. 2001. Distributing media transformation over multiple media gateways. In *Proceedings of the ninth ACM international conference on Multimedia, September 30 - October 05, Ottawa, Canada*, N. D. Georganas and R. Popescu-Zeletin, Eds. 159–168.

Opyrchal, L., Astley, M., Auerbach, J., Banavar, G., Strom, R., and Sturman, D. 2000. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In *Proceedings of Middleware*. 185–207.

Ozer, B. and Wolf, W. 2001. Video Analysis for Smart Rooms. In *Internet Multimedia Networks and Management Systems, ITCOM, Denver Colorado USA*. Vol. 4519. SPIE, 84–90.

Pietzuch, P. R. and Bacon, J. 2003. Peer-to-peer overlay broker networks in an event-based middleware. See Jacobsen [2003].

Pietzuch, P. R. and Bacon, J. M. 2002. Hermes: A distributed event-based middleware architecture. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*. IEEE Computer Society.

Pratt, W. K. 1991. *Digital Image Processing*. Wiley-Interscience. John Wiley & Sons, Inc.

Rafaelsen, H. O. and Eliassen, F. 2002. Design and performance of a media gateway trader. In *CoopIS/DOA/ODBASE*, R. Meersman and Z. Tari, Eds. Lecture Notes in Computer Science, vol. 2519. Springer, 675–692.

Segall, B., Arnold, D., Boot, J., Henderson, M., and Phelps, T. 2000. Content based routing with Elvin4. In *Proceedings of AUUG2K, Canberra, Australia*.

Smits, P. and Annoni, A. 2000. Cost-based feature subset selection for interactive image analysis. In *Proceedings of the 5th International Conference on Pattern Recognition*. Vol. 2. IEEE, 386–389.

Sun Microsystems 1999. *Java Media Framework, API Guide*, 2.0 ed. Sun Microsystems. http://java.sun.com/.

Taymans, W., Baker, S., Wingo, A., and Bultje, R. S. 2004. *GStreamer application development manual*, 0.8.8 ed. http://gstreamer.freedesktop.org/documentation/.

Terpstra, W. W., Behnel, S., Fiege, L., Zeidler, A., and Buchmann, A. P. 2003. A peer-to-peer approach to content-based publish/subscribe. See Jacobsen [2003].

Yang, J. and Honavar, V. 1998. Feature subset selection using a genetic algorithm. *Intelligent Systems 13*, 44–49.

Yang, M.-T., Kasturi, R., and Sivasubramaniam, A. 2003. A Pipeline-Based Approach for Scheduling Video Processing Algorithms on NOW. *IEEE Transactions on Parallel and Distributed Systems 14*, 119–129.

Zhou, W., Vellaikal, A., Shen, Y., and Kuo, J. C.-C. 1998. Real-time content-based processing of multicast video. In *Conference Record of the Thirty-Second Asilomar Conference on Signals, Systems and Computers*. Vol. 1. IEEE, 882–886.