Presentation Processing Support for Adaptive Multimedia Applications*

Edward J. Posnak, **Harrick M. Vin**, and **R. Greg Lavender**

Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188
E-mail: {ejp,vin,lavender}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885

ABSTRACT

This paper describes the design of and implementation of Presentation Processing Engines (PPEs) that provide flexible control over QoS to manage heterogeneity in networks, end-systems and compression formats. Development of PPEs is facilitated by a framework that provides the modular architecture, data types, and transformations common to most codecs, filters, transcoders etc. to facilitate the implementation of emerging compression standards and their integration into media processing applications. By allowing fine-grained composition of compression and image processing modules, the framework facilitates the development of extensible presentation processing engines that can be dynamically configured to adapt to changes in resource availability and user preferences.

1 Introduction

Recent advances in computing and communication technologies have made it economically viable to design and implement distributed multimedia information systems that promise to enhance users' ability to access a rich variety of audio, video, and textual information over globally inter-connected networks. Such systems will have to provide mechanisms for servicing clients over a wide range of heterogeneous computing and communication environments: ranging from hand-held devices to powerful workstations, and from the huge installed base of ethernets, token rings, telephone lines, to higher bandwidth and wireless networks. To manage this high degree of heterogeneity, sophisticated multimedia storage systems will maintain media objects at various levels of resolution, so that the access and delivery of the objects can be tailored to the computing and communication capabilities of client sites. Additionally, operating systems that support quality of service (QoS) guarantees will provide mechanisms that allow applications to adapt to fluctuations in the availability of computing and communication resources as well as to changes in quality desired by users.

To effectively utilize these mechanisms, and enable existing applications to access and process multimedia data stored in a variety of compressed formats, presentation processing mechanisms will be required to: (1) decouple applications from compression format by providing a uniform interface for accessing and processing data, (2) perform a number of common media processing operations (3) dynamically change the quality of presentation by appropriately adjusting decompression parameters, and (4) support new standards that emerge as compression technology evolves. Today's hardware codecs and digital signal processors lack the extensibility and configurability needed to meet all of these requirements. The design and implementation of a configurable software presentation processing mechanism that achieves these objectives is the subject

*This research was supported in part by IBM, Intel, the National Science Foundation (Research Initiation Award CCR-9409666), NASA, Mitsubishi Electric Research Laboratories (MERL), Sun Microsystems Inc., and the University of Texas at Austin.

matter of this paper.

Configurability is a key element in the design of presentation processing mechanisms. The first generation of internet conferencing applications, such as nv[4] and ivs,[18] were not built to be configurable, but rather to demonstrate their viability and facilitate experimentation. As a result, these systems suffer from lack of interoperability because they are tightly coupled to specific compression algorithms and chroma formats. Experiences gained from these systems and others have influenced the design of more flexible systems (such as vic,[7] Quicktime,[3] the VuSystem,[16] the Berkeley Continuous Media Toolkit,[12] etc.), which employ a configurable, modular approach that allows a variety of codecs to be supported. Such systems can be transparently configured to use different compression algorithms, allowing an application to access multimedia data independent of its compression format. Whereas these systems provide coarse-grained configurability at the codec level, we propose a finer-grained approach that facilitates the following key features :

- *Extensibility:* Fine-grained configurability allows modules from existing codec implementations to be suitably extended or reused to implement new codecs, thereby simplifying software development. Moreover, support for application level media processing operations (e.g. scale, clip) can be added by plugging in modules that implement these operations to a codec's internal implementation. The ability to perform these operations on semi-compressed, as opposed to uncompressed data, often yields a significant performance gain.[1,14]
- *Flexible QoS control:* By allowing basic codec building blocks (such as the discrete cosine transform (DCT), quantization, color conversion, etc.) to be dynamically configured, a system can provide flexible control over QoS parameters such as frame rate, spatial resolution, and Signal to Noise Ratio (SNR). Recent work on QoS filters has demonstrated that such flexible QoS control can be realized by incorporating fine-grained operations such as re-quantization, filtering of frequency coefficients, etc.[20] Whereas QoS filters are primarily targeted at the adaptation of compressed streams to cope with heterogeneity of networks, we apply these mechanisms at the client site to adapt to changing resource availability and user preferences.

This paper describes the design of a framework for implementing Presentation Processing Engines (PPEs) that employs fine-grained configurabilty to achieve extensibility and flexible QoS control. The framework provides the modular architecture, data types, and transformations common to most codecs, filters, transcoders etc. to facilitate the implementation of emerging compression standards and their integration into media processing applications. In contrast to monolithic implementations, adding support for media processing operations is greatly simplified by the capability to utilize fine-grained composition of media processing and compression modules to implement PPEs. A PPE transparently implements a variety of compression/decompression algorithms (including JPEG[19] and MPEG[5]) and provides applications with a uniform interface for processing multimedia data and controlling QoS parameters such as frame rate, spatial resolution, and SNR. The modular implementation of a PPE is configured at run-time according to the compression format of the media object, available resources, and the application's QoS requirements. Dynamic reconfiguration of a PPE provides the flexibility to control QoS throughout playback, and hence accommodate changes in resource availability and user preferences.

The remainder of this paper is organized as follows: Section 2 describes the design of a framework that facilitates the development of configurable PPEs. Section 3 illustrates the advantages of the fine-grained compositional approach to codec development using concrete examples of JPEG, MPEG and MPEG-2 decoder implementations. In Section 4 we compare the performance of our fine-grained modular approach to building codecs with corresponding monolithic implementations. Finally, Section 5 summarizes our contributions and comments on future work.

2 The Presentation Processing Engine

Media processing applications are naturally developed using a pipeline model, in which modules that perform various operations on the data are inserted between sources (e.g., microphones, cameras, etc.) and sinks (e.g., speakers, display devices, storage servers, etc.).[6,16,12] The PPE is a module that provides applications with a uniform interface for processing multimedia data and controlling the quality of presentation. The discussion that follows describes how our design addresses

three key issues:

- The development of new software codecs and their integration into media processing applications.
- The provision of flexible control over QoS parameters such as frame rate, spatial resolution, and SNR.
- The tension between static and dynamic composition that arises from the need for high performance implementations.

2.1 Codec Development and Integration

As compression technology continues to evolve, and coding formats proliferate, it becomes increasingly important for presentation processing mechanisms to add support for emerging compression standards. The PPE framework is designed to facilitate easy development of new software codecs and their integration into media processing applications by reusing architectural features, data types and transformations common to most codecs. The observation that many compression algorithms can be implemented as a sequence of transformations (e.g. DCT, quantization) on a media-specific data types (e.g. blocks, macroblocks) has motivated the development of a framework that enables these transformations to be implemented as composable modules.

The PPE framework provides a collection of reusable type templates that are parameterized and extended to define modules that implement compression operations (e.g. DCT, Huffman coding) and image processing functions (e.g. scale, clip). The templates provide mechanisms to compose modules by creating a data path from one module's output to another module's input. The sequence of data transformations that occur within a PPE is determined by the composition of its modules. Strong typing of modules, based on their input and output data types, is used to provide compile-time checking of correct module composition. Modules can be recursively composed of other modules to effectively capture the functional overlap that exists at multiple levels in codec implementations. For instance, whereas primitive level building blocks such as color conversion and DCT are widely used, more complex aggregate operations such as the DCT/quantize/zig-zag/run-length/Huffman-encode sequence are also common to many compression algorithms. Relatively large processing sequences can be reused in developing scalable codecs since each scalable profile is often a variation or extension of some base algorithm. Though minor differences in standards sometimes prohibits extensive code reuse, the development of new codecs can leverage off a reusable framework for module composition that can be parameterized with compression algorithm-specific code.

The integration of a new codec into applications requires the codec implementation to support the set of common media processing operations (translate, scale, clip, etc.) provided by the PPE. Applications invoke media processing operations by passing to the PPE a *Control Operation*, which is an object that indicates the operation to be performed and contains any relevant data. When the control operation is invoked on the PPE, a module to perform the operation is configured into the PPE's internal implementation, and continues to operate on the media stream until it is removed by another control operation. Many of the modules that implement these operations can realize significant performance gains by directly processing semi-compressed data. For example, the shrink-by-2 operation can be performed roughly five times faster on run-length encoded images than on images that are not compressed.[13] Control operations are also used internally by the PPE to determine the most efficient implementation of a service. For example, to support scaling, the PPE invokes a `Scale` control operation on the decoder, which attempts to reconfigure its modules to perform scaling in the frequency domain. If the decoder cannot support this control operation and returns a failure status, the PPE will then resort to a less efficient, spatial domain implementation. Because the PPE framework provides a uniform mechanism for composing modules, adding support for efficient media processing operations is greatly simplified. The operation can be implemented as a module and later configured into any codec's implementation to operate at the most efficient level possible. In contrast, integrating a new monolithic codec into existing applications requires each media processing operation to be reimplemented in the codec.

2.2 Flexible QoS Control

To enable applications to adapt to heterogeneous environments and changes in available resources and user preferences, mechanisms for controlling QoS parameters must be provided. For this purpose, the PPE provides the `SetQosLevel` control operation, which encapsulates requested values in terms of frame rate, spatial resolution, and SNR. The PPE handles this control operation by appropriately configuring its internal modules so as to meet the QoS requested. Modules developed within the PPE framework can be dynamically configured using an `Attach` method, provided in every module by a type template. When an application opens a media object, the compression format, available end system resources, and QoS requirements initially determine the module configuration in the PPE. When a change in QoS occurs, either due to client initiation or a change in resource availability, the PPE may need to replace internal modules with ones that produce different QoS characteristics and/or reconfigure its implementation to support a different scalable profile. Configuration changes are captured in a state machine where each state represents a different configuration of modules corresponding to some QoS level. The state transitions are associated with a set of operations that specify how to obtain the new configuration from the existing one, i.e. by creating, attaching, detaching, and deleting modules. A handler for the `SetQosLevel` control operation executes a state transition depending on the current and desired QoS levels. The ability to dynamically configure compression modules provides the application with more flexibility to control QoS than systems that rely solely on discarding parts of the bit stream.

2.3 Static versus Dynamic Composition Tradeoff

Object-oriented abstractions that enable a high degree of reconfigurability often come with an associated efficiency cost that results from interaction with multiple abstraction boundaries enforced for the sake of information hiding and a high degree of modularity.[2,17] The performance sensitive nature of software presentation processing requires a carefully engineered balance between the use of static and dynamic composition of modules. Since static binding enables a compiler to make efficient choices during code generation (e.g., inlining), whereas dynamic binding often results in less efficient run-time actions (e.g. dynamic method lookup and dispatch), the PPE framework provides mechanisms that allow modules to be composed in either fashion. The statically determined components of an implementation are represented by a set of parameterized types that allow an efficient coupling of modules that require high performance and have minimal reconfigurability demands. The ability to define parameterized types allows for modular composition, but the efficiency cost is minimized by the use of inline methods, which effectively collapse a sequence of transformations into highly efficient code. Whereas inlining in general does not always result in better performance, empirical evidence suggests that careful selection of which methods are inlined can have a positive benefit on performance. This optimization is very effective for bit stream filters and Huffman decoders where performance is critical and dynamic configurability is of little value. The PPE framework makes the same kind of abstraction tradeoffs that are evident in the ANSI C++ Standard Template Library (STL),[8,15] which is a model for how to preserve syntactic abstractions that balance the tension between static and dynamic binding choices while incurring minimal run-time overhead.

3 PPE Composition: Examples

Our fine-grained, compositional approach to implementing PPEs facilitates the development of new software codecs and their integration into media processing applications. Additionally, the ability to dynamically configure codec components provides increased flexibility in controlling presentation quality. To demonstrate the effectiveness of our approach, we have developed a library of configurable modules and have used it to implement a variety of codecs. In what follows, we describe the construction of JPEG, MPEG and MPEG-2[11] decoders using the PPE framework. The example JPEG decoder illustrates the process of module composition and how the framework simplifies the insertion of media processing operations into a PPE configuration. The MPEG example demonstrates the reusability of the architecture and illustrates how module configurability is used to perform late binding, and provide flexible control over QoS. The MPEG-2 example demonstrates how a state machine can be used to dynamically reconfigure the codec implementation to support different scalable profiles.

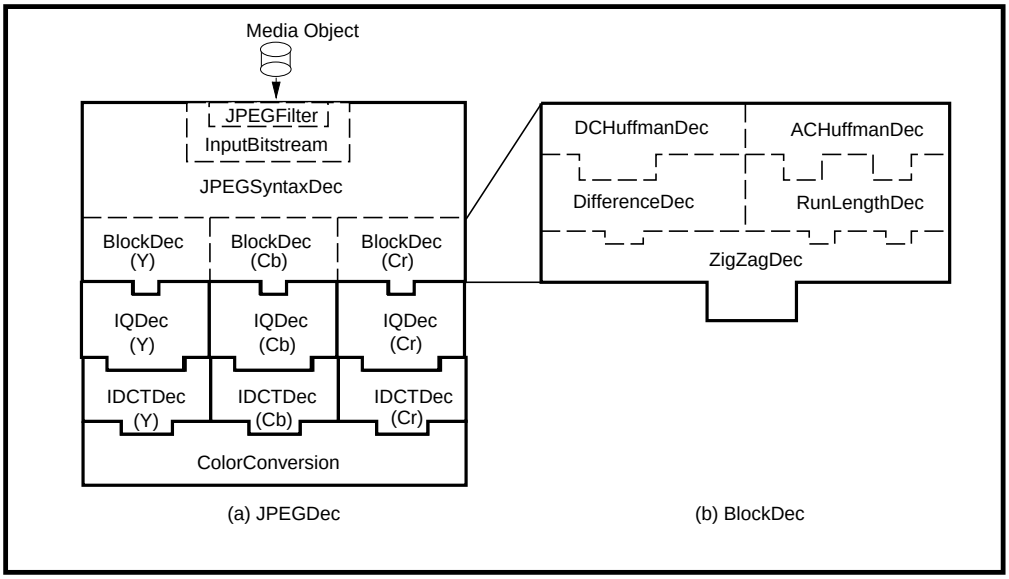

Figure 1 : Module Composition for a JPEG Decoder. (a) JPEG Decoder (b) block decoder module. Different shapes are used to indicate different module types. Modules dynamically bound via Attach are shown with a bold outline. Internal, statically bound modules are shown by a dashed outline.

3.1 JPEG Decoder

JPEG is a widely used standard for still image compression. A JPEG encoder fragments an image into 16x16 pixel blocks, called macroblocks, that are further divided into a number of 8x8 pixel blocks for different color planes such as Y, Cr, and Cb. The 8x8 blocks are separately transformed into the frequency domain using the discrete cosine transform (DCT). The resulting frequency coefficients are quantized to filter out less visually perceptible components of the signal and then scanned in zig-zag fashion to obtain an approximate ordering from lowest to highest frequency. The lowest frequency DC coefficient is difference encoded against the DC value in the previous block of the same color plane and the remaining AC coefficients are run-length encoded. Finally, the blocks are further compressed using an entropy coding algorithm such as Huffman encoding. The decode path performs the inverse of these functions in the reverse order.

Figure 1 illustrates how a JPEG decoder is composed from various PPE library modules. Input from the media object is parsed by the syntax decoder and separated into control data (e.g. frame size, chrominance subsampling, quantization tables, etc.) and actual coded image data. Control data is used to set decoder state variables and to determine module configurations. The coded image data is passed down pipelines of modules, which reconstruct the image for display. The first module in each pipeline, the block decoder, reads variable length coded coefficient data from the input bitstream and performs the Huffman, run-length, difference, and zig-zag decoding operations to reconstruct an 8x8 block of coefficients. These coefficient blocks are further transformed by the inverse quantizer module, the inverse DCT module, and are finally aggregated in the color conversion module, which reconstructs the image for display. The modules in a pipeline are shown having different shapes that fit together to illustrate the fact that strong typing is used to ensure that only feasible configurations are permitted. For example, the types of 8x8 blocks passed between the block decoder, inverse quantizer, inverse DCT and color conversion modules are purposely made distinct to prevent misconfigurations at compile time.

Efficient and flexible application of media processing operations is facilitated by the modular architecture. For example, the use of separate pipelines for the different color planes allows efficient brightening or darkening of an image or region to be easily performed by inserting a module that alters the DC coefficient of luminance (Y) blocks. Other operations such as requantization and pixel subsampling, which are luminance/chrominance specific, can be implemented by inserting modules in the proper pipelines. Such flexibility facilitates the implementation of adaptive QoS filters and transcoders.

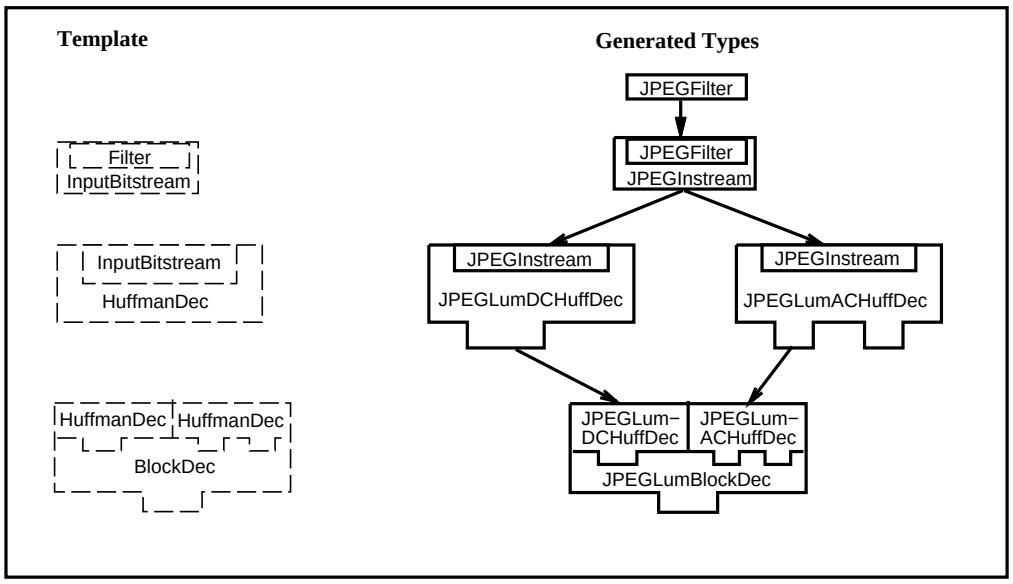

Figure 2 : Module composition using parameterization to generate a various types of input bitstreams, Huffman decoders and block decoders.

The block decoder, shown in Figure 1(b), exemplifies how static module parameterization is used to achieve high performance while maintaining modularity. The actual Huffman decoders used for a given block are context dependent (i.e. they depend on whether the block is luminance, chrominance, intra-coded, etc.). While it is possible to dynamically reconfigure a block decoder's internal modules to handle these cases, this approach is grossly inefficient. Consequently, inlined interface extensions and module parameterization are employed to effectively collapse the internal pipeline into a high performance, statically bound configuration. A number of block decoder types (e.g. intra-luminance, intra-chrominance, etc.) can be created by parameterizing the block decoder with a pair of Huffman decoders that provide inline methods for decoding DC and AC coefficients, respectively. Figure 2 shows graphically how different types of block decoders are generated using module parameterization. The templates on the left side of the figure are used to generate specific module types, which are shown on the right. The figure illustrates how a JPEG luminance block decoder is composed of luminance AC and DC Huffman decoders, each of which is parameterized by an input bitstream. The input bitstream itself is parameterized by a codec-specific filter that handles codec-specific differences in bytestream syntax (e.g. zero stuffing in JPEG, AAL headers in DSJ[10]). Factoring out the common code relieves the developer from having to implement, debug, and optimize a new bitstream object to support or experiment with a new bytestream syntax. This approach, when combined with selective inlining of performance critical functions, maintains a highly configurable architecture whose abstraction boundaries are enforced at compile time, but compiled away to produce efficient code.

This example has shown how the PPE framework facilitates the development of decoders that can easily integrate media processing operations and how static and dynamic module composition are used to achieve flexible, efficient implementations. Encoders, transcoders and QoS filters can be developed within this framework using a similar methodology. The next example shows how, due to structural similarity between codecs, we can reuse much of the design to construct an MPEG decoder.

3.2 MPEG Decoder

The MPEG compression algorithm exploits the large temporal and spatial redundancies present within an image to achieve a high degree of compression.[5] MPEG supports three kinds of frames: intra (I), forward predicted (P), and bi-directionally interpolated (B). I frames are compressed in a fashion similar to JPEG and are used as references for coding

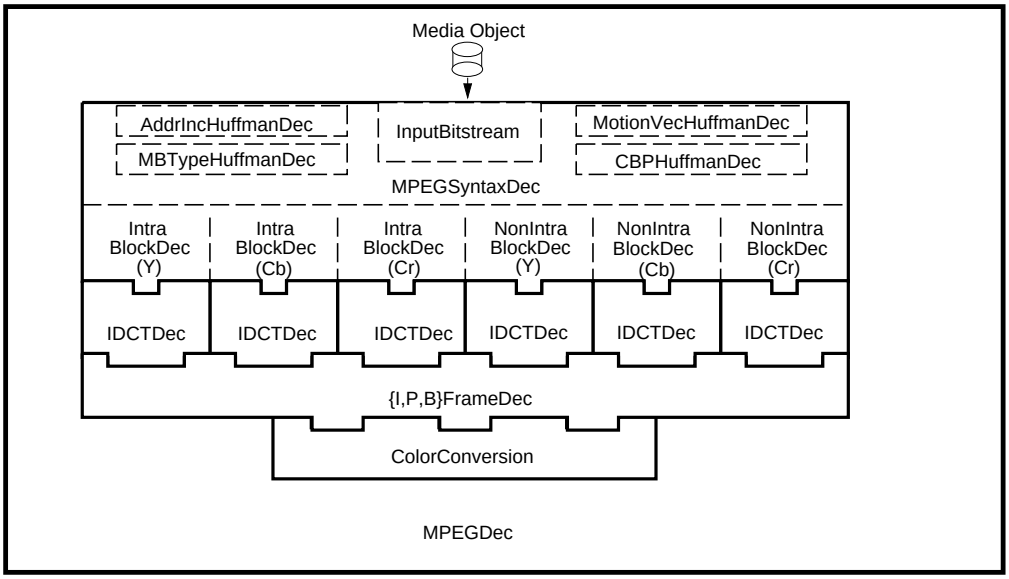

Figure 3 : Module Composition for an MPEG Decoder

of P and B frames. Macroblocks in P frames may be intra coded or coded as differences from some macroblock in the previous reference frame, whose relative location is given by a motion vector. B frame macroblocks may be intra coded, predicted from either a past or future reference frame using motion vectors, or interpolated from both. Regardless of the type of frame it belongs to, each macroblock is partitioned into six 8x8 pixel blocks – four luminance and two chrominance blocks. Each of these 8x8 pixel blocks are transformed into the frequency domain using discrete cosine transform (DCT). The motion vectors in the P and B frames are difference and entropy encoded in a lossless manner. Significant compression gain is achieved when P or B frame macroblocks are highly correlated with their reference frames and can be either partially coded (using a coded block pattern), coded only as motion vectors, or skipped completely (using a macroblock address increment).

The architecture for an MPEG decoder, shown in Figure 3, is similar to the JPEG decoder in the previous example. The huffman decoders, block decoder, bit stream, inverse DCT, and color conversion modules are configured in much the same way, passing many of the same data types (e.g. blocks, macroblocks etc.). Again the input bitstream is parsed by the syntax decoder and separated into control data and coded image data. Huffman decoder modules for address increment, macroblock type, motion vectors and coded block pattern are used to parse encoded control data. The block decoder module template has been parameterized to create new MPEG-specific block decoder types for intra and non-intra coded, luminance and chrominance blocks. To gain further efficiency these block decoders are also parameterized with a dequantization module, so that dequantization/oddification/saturation is performed only on coded coefficients. The MPEG decoder pipeline includes a frame decoder that performs temporal prediction and motion compensation. There are three types of frame decoders, one each for I, P, and B frames, that are dynamically attached as each new frame type is parsed from the input stream. This is done merely to simplify the implementation, since the overhead of reconfiguring every frame is relatively low.

Dynamic configurability is used to allow late binding of modules. For example, to support different display types, the color conversion module must be able to attach to modules that accept different data types(e.g. colormapped, 24-bit RGB, monochrome, etc.). When the output is a window, the output type is platform dependent and cannot be determined until run time. The color conversion module provides a control operation to set its output type, which can be invoked when the PPE attaches to the sink and the output type becomes known. When this control operation is handled the color conversion module binds its implementation based on the target output type.

A primary benefit of configurability is to achieve flexible control over QoS. For example, the color conversion module

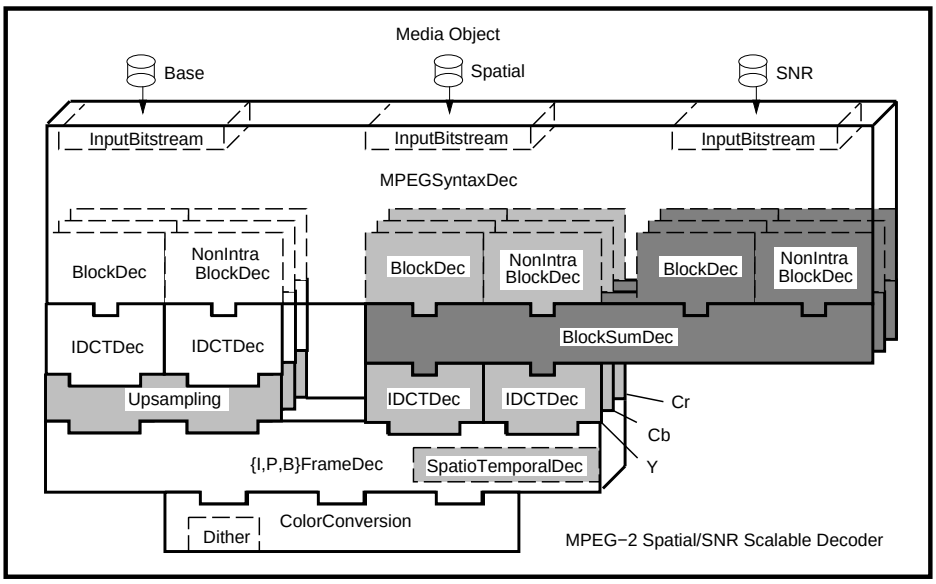

Figure 4 : Module Composition for an MPEG2 Spatial/SNR Decoder. Y, Cb, and Cr pipeline instances are shown in 3 dimensions. Light shading indicates modules that are attached to support spatial scalability, dark shading indicates modules that are attached to support SNR scalability.

may employ dithering to remove banding effects and improve the quality of the image. However, various dithering algorithms can be employed to achieve different frame rate and SNR characteristics.[9] Control operations are provided to allow the color conversion module to dynamically change the dithering algorithm from a high quality slow dither, to an average quality fast dither, or to no dither. This use of configurability can be applied to the inverse DCT and other modules to provide more control over the frame rate and SNR characteristics of a video stream during playback.

3.3 MPEG-2 Decoder

The MPEG-2 standard supports a number of scalable profiles that allow a video stream to be decoded at multiple quality levels. In the spatial scalable profile, frames are encoded at two levels of spatial resolution. The base layer provides a low-spatial resolution image that can be decoded independently, or upsampled and combined with enhancement layer data to produce a higher resolution image. A macroblock in the enhancement layer may be predicted from the base layer, from a previous full resolution reference frame using motion compensation, or may be a weighted sum of both predictions. In the SNR scalable profile the base and enhancement layers code the same images at different levels of quantization. The base layer may be decoded independently or combined with enhancement layer data to produce a higher quality image. The hybrid Spatial/SNR scalable profile allows decoding at three quality levels using base, spatial enhancement, and SNR enhancement substreams. At the lowest quality level, the base substream is decoded independently. The base and spatial enhancement streams may be combined as in spatial scalable mode to produce a medium quality image. To achieve the highest quality image, the spatial and SNR enhancement streams may be added as in SNR scalable mode and then combined with the base layer as in spatial scalable mode.

A decoder that supports all three quality levels has three associated states, which we shall call base, spatial and SNR. Figure 4 abstractly shows the configuration of a decoder in the highest quality, or SNR state. Many of the modules and configurations are similar to the decoders described in the previous examples. Since the spatial scalable profile uses temporal and spatial prediction, a module has been added to the frame decoder to perform these operations. This module combines macroblocks decoded from base and spatial substreams to produce high resolution macroblocks. The additional modules

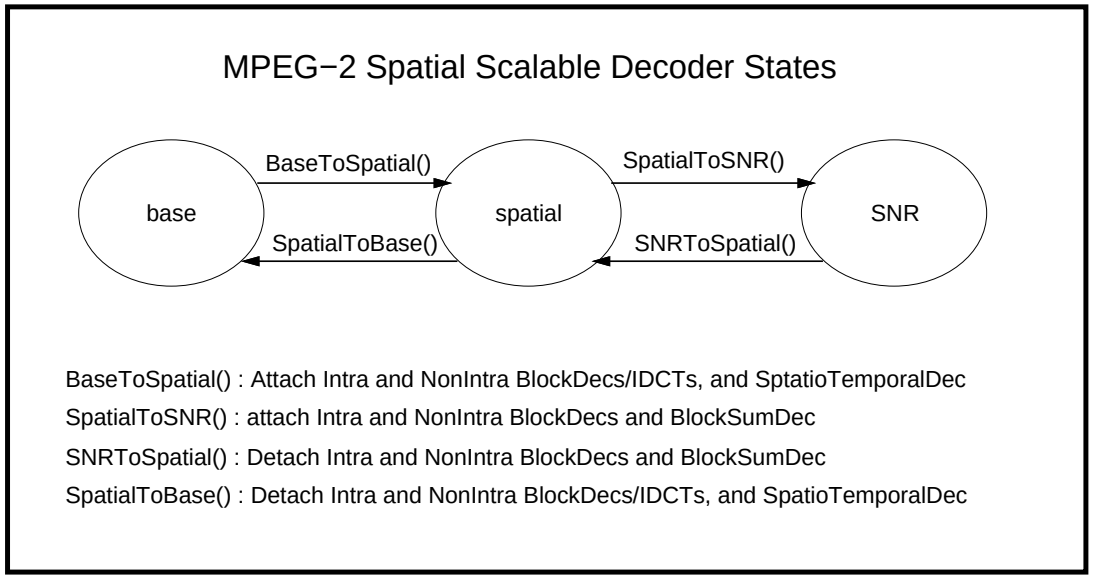

Figure 5 : State Machine for MPEG-2 Spatial/SNR Scalable Decoder

needed for spatial scalability are shown with light shading in Figure 4. The SNR substream contains fine quantization data that must be added to the spatial enhancement layer before converting transform coefficients back to the spatial domain. The block summing module is used to add blocks of quantized coefficients from the spatial and SNR substreams and output this sum to the inverse DCT module for the spatial substream. This module can be inserted between the block decoders and the IDCT since it accepts quantized frequency coefficient blocks as input and produces them as output. The additional modules needed for SNR scalability are shown with dark shading in Figure 4.

When a change in QoS requires switching between resolution levels, the protocol machine changes its configuration by adding, deleting and attaching modules. Figure 5 shows the decoder states associated with each resolution level and the configuration operations associated with each state transition. To go from high to medium resolution, the decoder transitions from SNR to spatial state by deleting the module sequence attached to the SNR substream and the block summing module. To switch to the base state, the module sequence attached to the spatial substream, the spatio-temporal prediction module and the upsampler are removed. The frame decoders are replaced by the MPEG-1 frame decoders, which perform base layer temporal prediction and motion compensation.

The examples in this section have borne out the following benefits of our fine-grained modular framework for composition of PPEs:

- The development of new codecs and their integration into media processing applications is greatly simplified by a domain-specific framework that facilitates design reuse. Construction of JPEG, MPEG, and MPEG-2 decoders is accomplished by parameterizing the same basic architecture with compression-specific algorithms, rather than building each decoder from scratch.
- Flexible control of QoS parameters such as frame rate and SNR is made possible by the ability to (1) dynamically configure modules (e.g. inverse DCT, dithering) that produce different QoS characteristics. and (2) switch between scalable profiles by reconfiguring the decoder according to predefined state transitions.

Image	Dim.	B/P	PPE	IJG
apollo	640x400	1.06	0.284	0.275
astrnaut	523x478	1.13	0.269	0.259
lena	512x512	1.33	0.311	0.300
mars	727x737	0.92	0.566	0.554
mirri3	551x579	1.90	0.425	0.408

Sequence	Dimensions	B/P	PPE	UCB
bike	352x240	0.41	13.4	14.8
flowg	352x240	1.78	7.9	7.7
jfk	192x144	0.79	31.6	30.7
tennis	352x240	0.79	11.2	11.6
zoom	256x192	0.64	20.8	20.1

Table 1 : (a) time to decode a JPEG frame for PPE and IJG JPEG decoders (b) frame rate achieved by PPE and UCB MPEG players. Dimensions (Dim.) and bits per pixel (B/P) are given for all media objects.

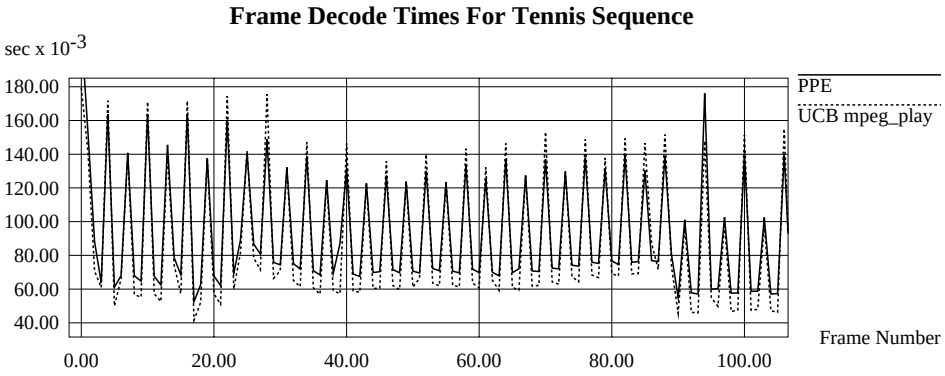

Figure 6 : Time to decode each frame for PPE and mpeg_play

4 Performance Evaluation

Given the qualitative benefits of the fine-grained compositional approach to building PPEs, a quantitative performance analysis is essential to demonstrate the viability of this approach. We have implemented several PPE-based codecs for this purpose, and here we report some early performance results. The JPEG and MPEG codec implementations described in the previous section are compared with the Independent JPEG Group's (IJG) JPEG decoder release 6, and University of California at Berkeley (UCB) MPEG player.[9] All experiments were performed on a Sun Sparc-20 system with no other user-level processes running.

For the JPEG experiments we used five different images, described in Table 1(a), to cover a wide range of image sizes and compression ratios. In each experiment the same image was decoded 100 times and the time to decode each frame was measured. The IJG decoder was run with high performance options turned on - i.e. a fast IDCT that performed dequantization, a fast upsampling algorithm, and a color conversion algorithm optimized for the particular chrominance subampling. These same optimizations were also used in the PPE implementation. The minimum time to decode each image, by the PPE and IJG implementations is shown in Table 1(a). For the MPEG comparison, five test sequences were used, and each was decoded ten times. Both the UCB mpeg_play decoder and our implementation were configured to optimize for high frame rate by employing ordered dithering, average cheating in bidirectional interpolation, and a fast integer DCT algorithm. To eliminate display overhead from the comparison, all images were reconstructed and dithered but not sent to the X-server for display. The highest frame rate observed for each decoder and sequence is shown in table 1(b). A frame by frame decoding time comparison for part of a representative sequence is shown in Figure 6.

For all the JPEG sequences, the time taken to decode a frame by the PPE implementation is within 5 percent of the time required by the IJG implementation. Similarly, for MPEG sequences, the PPE frame rates are at most 10 percent lower than those of mpeg_play. These experiments indicate the performance of codecs developed within the modular PPE framework is comparable to that achieved by monolithic implementations.

5 Conclusion

We have described the design and implementation of a presentation processing engine that provides flexible control over QoS parameters to allow applications to adapt to heterogeneous environments, fluctuations in resource availability, and changing client demands. We have shown, through examples, how the modular framework facilitates the development of efficient PPEs that can dynamically bind and reconfigure their implementations to transparently provide applications with these capabilities. Our performance evaluation indicates that a modular approach to developing to PPEs is viable.

Based on lessons learned from our implementations, we plan to examine how compression standards might be designed with software, rather than hardware, implementation as a first concern. The PPE framework will be used to develop experimental codecs to support this research. We also plan to investigate how applications can use the flexibility of modular composition to improve transmission performance and reliability by integrating transport level functions into the PPE. Given that compression technology is still evolving and presentation processing is the current bottleneck in communications performance, improvements in this area should have a significant impact on future distributed multimedia applications.

6 Acknowledgements

We would like to thank Sriram Rao, Pawan Goyal, and the anonymous reviewers for their helpful feedback on the earlier version this manuscript. We would also like to acknowledge Scott Page for his input during numerous discussions on configurable frameworks and Prashant Shenoy for his editorial suggestions, which significantly improved the paper. Finally, we are grateful to Paul Jardetzky for shepherding this paper.

7 REFERENCES

[1] Elan Amir and Steve McCanne. An Application Level Video Gateway. In *Proc. of ACM Multimedia '95*, November 1995.

[2] David D. Clark. Modularity and Efficiency in Protocol Implementation NIC-RFC 817. In *DDN Protocol Handbook*, pages 3.63–3.88. U.S. Department of Defense, July 1982.

[3] E. M. Hoffert et al. QuickTime: An Extensible Standard for Digital Multimedia. In *Proceedings of IEEE Compcon'92*, pages 15–20, 1992.

[4] Ron Frederick. Experiences with real-time software video compression. In *Sixth International Workshop on Packet Video*, July 1994.

[5] D. Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):47–58, April 1991.

[6] S. Gibbs. Composite Multimedia and Active Objects. *OOPSLA '91*, pages 97–112, 1991.

[7] Steve McCanne and Van Jacobson. vic: A Flexible Framework for Packet Video. In *Proc. of ACM Multimedia '95*, November 1995.

[8] D.R. Musser and A.A. Stepanov. Algorithm-Oriented Generic Libraries.

[9] K. Patel, B. C. Smith, and L. A. Rowe. Performance of a Software MPEG Video Decoder. In *Proceedings of ACM Multimedia 93*, Anaheim, California, August 1993.

[10] E. J. Posnak, S. P. Gallindo, A. P. Stephens, and H. M. Vin. Techniques for Resilient Transmission of JPEG Video Streams. In *Proceedings of Multimedia Computing and Networking, San Jose, CA*, February 1995.

[11] A. Puri. Video Coding using the MPEG-2 Compression Standard. *Proceedings of SPIE Visual Communications and Image Processing*, 2094:1701–1713, Nov 1993.

[12] L.A. Rowe. Continuous Media Applications. Presented at Multipoint Workshop held in conjunction with ACM Multimedia '94., November 1994.

[13] B. Smith. Fast Software Processing of Motion JPEG Video. In *Proceedings of the ACM Multimedia'94*, pages 77–88, October 1994.

[14] B. Smith and L. Rowe. Algorithms for Manipulating Compressed Images. *IEEE Computer Graphics and Applications*, pages 34–42, 1993.

[15] Alexander Stepanov and Ming Lee. The Standard Template Library. Technical report, Hewlett-Packard Laboratories, 1995.

[16] D. L. Tennenhouse and et. al. A Software-Oriented Approach to the Design of Media Processing Environments. In *International Conference on Multimedia Computing and Systems*, volume 1, pages 435–444, Boston, Massachusetts, May 1994.

[17] David L. Tennenhouse. Layered Multiplexing Considered Harmful. In Harry Rudin and Robin Williamson, editors, *Proc. IFIP WG6.1/WG6.4 International Workshop on Protocols for High-Speed Networks*, Zurich, Switzerland, May 1989.

[18] Thierry Turletti. The INRIA Videoconferencing System IVS. *Connexions*, 8(10):20–24, October 1994.

[19] G. K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):31–44, April 1991.

[20] Nicholas Yeadon, Francisco Garcia, Doug Shepherd, and David Hutchison. Continuous Media Filters for Heterogeneous Networking. In *Proceedings of Multimedia Computing and Networking, San Jose, CA*, January 1996.