An Open Software Architecture for Structured Data Elaboration and Transcoding

Luca Vollero, Giulio Iannello
Dipartimento di Informatica e Sistemistica
University of Naples, Federico II
{*vollero, iannello*}*@unina.it*

Francesco Delfino
ITEM - Laboratorio Nazionale CINI
Napoli
fdelfino@consorzio-cini.it

Abstract

In this paper, we propose a software architecture model for the development of elaboration/transcoding modules for structured data. We define a flexible implementation approach which makes it easy to generate modules that deliver good performance. The solution has also the objective to foster high reusability and independent coding of processing and I/O modules. A set of modules developed using this approach have proved to deliver good performance and flexibility in processing MPEG2 Video streams. This paper reports measurements and comparison results with a standard MPEG2 Video decoder, that confirms the potentiality of the approach.

Keywords: Transcoding, real-time processing, software modularity, code reusability.

1 Introduction

The need to quickly meet the new trends in digital data processing, such as the adaptation of existing algorithms to new formats and scenarios, calls for fast implementation methods, such as using pre-built, parametric modules. These modules should allow the changes required to address specific problems to be minimized. Moreover, there is a need to adopt a general and an efficient framework, which is a key target for network-based real-time applications.

In this paper we present an open software architecture for elaboration and transcoding of digital structured data, with the objective of creating a reference for quick module development, following a layered approach. In addition, we propose a method for pratical implementation of these modules and effective communication among them.

We based our approach on exploiting the natural organization of structured data, and on defining the data processing modules accordingly. The advantage of this approach is the ability to sistematically arrange and select the modules that are directly involved in a specific transformation.

Moreover, using pre-built modules and modifying just one module to accommodate a specific requirement minimizes software development. Obviously, keeping the development activity to a minimum enforces reliability and minimizes the efforts on performance optimization of the whole elaboration/transcoding process.

The organization of the paper is the following. In section 2 we discuss how the structure of digital data can be used to organize data processing. In section 3 we review previous works about the transcoding-elaboration problem. In section 4 we propose a general architecture to implement eloaboration modules. In section 5 we propose an effective implementation of this architecture. Finally, in sections 6 and 7 we present some experimental results and conclude the paper.

2 Structured data organization and processing

Digital data is often structured and compressed to minimize storage space and transmission time. First, the raw data is analyzed for macro-information detection; then macro-information levels and data are mixed to build the structured data stream. The macro-information is often composed of multiple levels, because different levels capture different structural elements of the raw data type. As an example we can consider video data; in this case a typical data structure organization is depicted in figure 1.

Using structured and compressed data offers at least two advantages: (i) compression reduces space and bandwidth requirements, and (ii) structuring makes the data easier to process. The compression stage adds some processing overhead, but it is a necessary step for a useful structuring scheme. Typically the compression stage follows the structuring step, because the latter is used to limit the amount of data to be compressed simultaneously reducing the overall

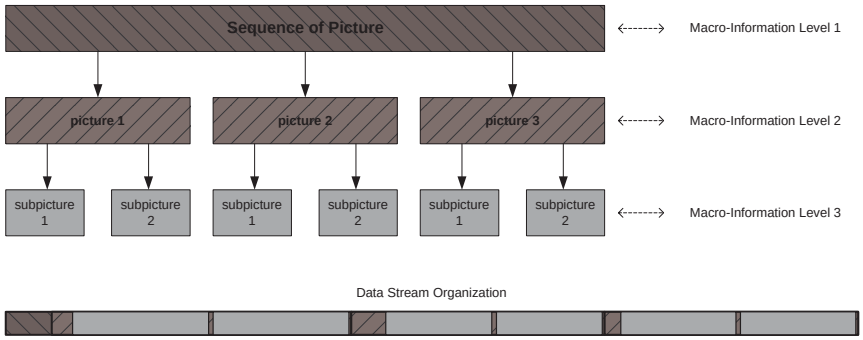

Figure 1. Typical Structure Organization for Video Data.

computational load.

A simple approach to process (compressed) structured data is to define two levels of analysis, the first one responsible for extracting information from the embedded macro-level structures, and the other responsible for the data management at the finest granularity. Doing so makes the processing method completely generic, and a single implementation method can be used for all problems. The flexibility of this scheme allows for good quality elaborations, but the complete analysis of the embedded data structure needs a computational effort usually not suitable for real-time time constrained operations, such as the case of on-line multimedia data bit-rate adaptation.

To improve efficiency, the embedded organization of (compressed) structured data can be exploited. The main advantage of this method is the ability to select the macro-information level to which apply the data processing.

When elaboration is based on data at macro-information level n, we can consider data at macro-information level n+1 as an atomic unit. Since the overall computational load is limited by the level of granularity at which transformations can take place, the trade-off between quality and complexity can be easily controlled.

3 Related work

Elaboration and transcoding of structured digital data is an ongoing effort, and the literature offers many references of the work being done. In [1], McCanne *et al.* propose an application level gateway for the transcoding of digital video data. The goal of this work is to scale the bandwidth requirements to accommodate users with bad connection and/or with reduced computational power. In [2], Khan and Yang propose an architecture for parallel transcoding of Mpeg Video compressed data; they use the independence of the macro-structures to parallelize the operation.

In [3], [4] and [5], Youn *et al.* investigate the transcoding problem for motion compensated video data streams, focusing on temporal dependencies and optimization of the inter-picture information reuse. In these works we have an example of correlated macro-information structural level transformations, where the changes of inter-picture information follows from the changes of picture format. In [7], Sun, *et al.* examine the Mpeg Video scaling problem, whereas in [6] Youn, Sun and Lin examine the possibility to parallelize the output stage of a transcoding module with one source and multiple sinks using once the source macro-information structure.

In [8] Smith and Rowe analyze the direct elaboration of compressed JPEG pictures; the goal is to apply a transformation directly over a coded picture. In [9] Smith and Acharya examine the direct elaboration of a compressed MPEG Video stream. This problem is more complex than the first one for the double macro-information level dependencies: spatial and temporal.

From these, as well as from other solutions, we can define three general models for the elaboration-transcoding applications.

The simplest approach to the problem is to come back to a raw data format, apply the desired elaboration computing process, and then compress again the transformed data ([1]). This approach is depicted in figure 2; the main problem of this solution is poor performance, which usually bans from using it in real-time scenarios.

Typically the encoding stage is where the performance is critical; a work-around solution, used in [6], simplifies this step, as depicted in figure 3. The main idea is that a lot of information useful for the encoding process is already present in the encoded data stream; this data can be reused or, accordingly, it can direct the encoding process [1].

A third approach to the problem, used in [8] and [9], is shown in figure 4. This solution tends to simplify all the

[1] For example, the motion compensation in video data compression can use the encoded information to direct the correlation step of the encoding stage, reducing complexity and processing times.

Figure 2. The Decoding-Processing-Encoding Scheme.

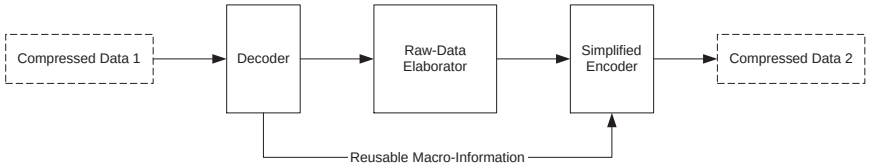

Figure 3. The Simplified Decoding-Processing-Encoding Scheme.

decoding-encoding stage, and to concentrate all processing in one step. Is so doing we trade-off diminished flexibility with increased performance. Flexibility is reduced because data processing must follow the structure of compressed streams, which prevents, for instance, from translating substantially different compression formats. Conversely, similarities between input and output formats may be exploited to improve the processing efficiency.

4 Architecture

In the previous section we described three methods to elaborate/transcode digital structured data types. However, only the simplified decode-elaborate-encode and the direct elaboration methods are of practical interest in on-line scenarios. This is the main reason why we focus on them to define a general software architecture for digital structured data elaboration.

The architecture assumes that the computational granularity is fixed, i.e. the transformation works on the data stream down to a given macro-information level; in this paper, the parameter determining this level will be referred to as the *operational-line*. Starting from this assumption, we establish two levels of analysis of the structured data. Above the defined macro-information level we can manipulate and transform all objects and structures, while below this line we cannot process data, but simply transfer data blocks from the input to the output streams, possibly rearranged if required by the output formats. This organization gives us a mean to keep as low as possible the computational load associated to the transformation.

From an architectural point of view, the definition of the operational-line can be arbitrary. However its placement become a critical point when a specific instantiation of the architecture is considered. Conceptually, the architecture draws the *operational-line* at a level where a structured object becomes atomic, hence unchangeable. Any structured data transformation happens above that level, and it's bad placement can constrain transformation capabilities or can introduce unnecessary processing overhead.

Once the operational-line has been fixed, the elaboration can be performed as follows: first we parse the data to find the high level semantic structures, then we use this kind of information to guide the data processing/transcoding process. The structured data extracted from this parsing process can then be used to build several kinds of processing modules. For instance we can combine multiple macro-information layers such as picture degradation and frame skip in a video sequence.

These steps are the key elements of the transformation, but there is another task that we must consider to define a more general processing architecture: the I/O access patterns. Indeed, making the architecture independent of the mechanisms used to get inputs and produce outputs enables reuse over a wide range of I/O subsystems. In order to reach this goal, we separate data processing from physical I/O, adding to the architecture a layer to map virtual I/O operations to physical ones.

A complete architecture that fulfills all those requirements is depicted in figure 5. The architecture involves three layers. The higher extracts the semantic level information useful for the elaboration process. The middle one defines the specific task, combining atomic data blocks. The last one provides I/O interfacing for the module that we want to build.

The resulting architecture defines a general framework; fixing the data format and the *operational-line* we obtain a family of transformation parametrized with respect to physical I/O devices. The final module is then obtained by implementing the I/O layer.

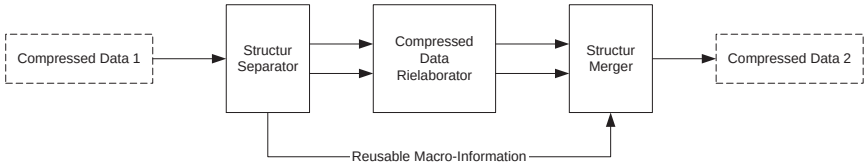

Figure 4. The Direct Elaboration Scheme.

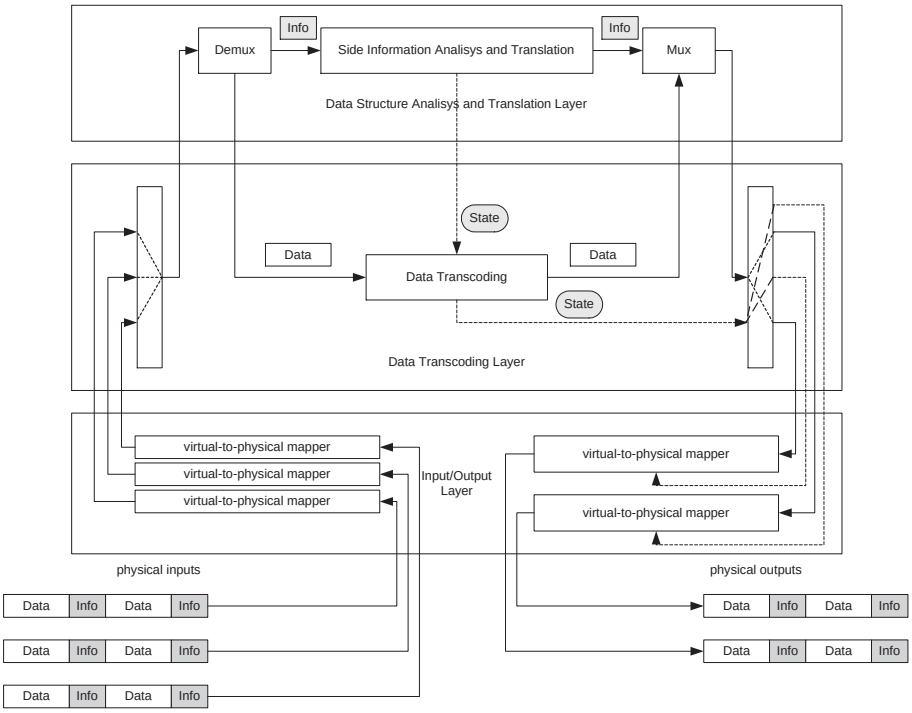

Figure 5. Complete Software Architecture.

5 Implementation

In this section we propose an implementation strategy for the architecture presented in the section 4. Our goal is to obtain a good trade-off between software modularity, processing flexibility and performance. We chose to use C++ and its object-oriented mechanisms: polymorphism and generics. Using these mechanisms the proposed solution resembles the modularity of the architecture and allows high performance with good flexibility and reusability (see section 6 for further details).

The polymorphism is used this way: we can think of the elaboration/transcoding module *like-a* specialization of the parsing module. The latter captures the main structures from the input stream and, possibly, translating them to the output stream. The former adds the ability to apply data handling capabilities. A specific elaboration module is derived from the abstract parsing class simply by redefining a bounded set of class methods. The information extracted by the parser can then be easily accessed by the computing part using inheritance and protected member variables.

The generics mechanism is used to define the abstract interface between the computing part on one side, and the data sources and destinations on the other side. This interface can then be implemented writing reusable I/O classes.

The resulting implementation approach is so based on a *Scalable Generic* Application Framework. The *scalability* is obtained defining the granularity of the parsing process, i.e. the *operational-line*; the *generic* means the ability to implement the processing layer abstracting from the I/O. The application framework is characterized by two main virtual methods: the former starts the parsing process at the desired granularity, the latter accesses the streams through *generic* interfaces and can be redefined to perform the de-

Module Name	Input	Output	Aim
RTPPacketizer	*File*	*Network*	*Organizing the input stream into an RTP packets sequence.*
RTPClient	*Network*	*File*	*Reconstructing the input stream from an RTP packets sequence.*
Degrader	*File* *File* *Network* *Network*	*File* *Network* *File* *Network*	*Reducing the transmission rate; interfacing file and/or network source/destination.*
Splitter	*File* *File* *Network* *Network*	*2-File* *2-Network* *2-File* *2-Network*	*Splitting the input stream into two output streams. The first one is a reduced version of the input stream.*
Merger	*2-File* *2-File* *2-Network* *2-Network*	*File* *Network* *File* *Network*	*Merging input streams and restoring the high quality level stream.*

Table 1. Implemented Modules

sired transformation from inputs to outputs.

Using this implementation strategy, we developed a set of MPEG2 Video elaboration modules, interfaced with multiple I/O pairs. The list of all these modules with a brief description is in table 1. The network interface (second and third columns in the table) consists of an active process, asynchronous with the elaboration/transcoding one, that extracts a bitstream from a packetized stream and vice versa. In general, the elaboration modules can be interfaced with I/O modules capable of complex tasks, like specialized scheduling algorithms and robust schemes of transmission.

To point out how this implementation strategy makes the development of new modules easy, we briefly give some details about the *Degrader*. The parsing process, common to all modules in table 1, assigns input information to two categories: (i) all video structures and significant data, and (ii) information that could be dropped, introducing controlled degradation in the video final quality[2]. The assignment is made available through a `state` variable of the parsing virtual class. To implement the *Degrader*, we only have to redefine the bitstream access method of this class. In pseudo-code, the new method's body looks like the following:

```
.....
// input-output bitstream declaration
T_in *bs_in;
T_out * bs_out;
.....
template<class T_in,class T_out>
unsigned MPEGDegrader<T_in,T_out>::
    parsing(int num, unsigned value){
        if(state >= significant){
            return bs_out->putbits(int num,
                       bs_in->getbits(num));
        }else{
            return bs_in->getbits(num);
        }

}
.....
```

The `significant` attribute determines the data that have to be preserved in the input stream, which are raw copied from input to output, whereas data are just read to push ahead the parsing process and then dropped from the output, when they are less than `significant`. Similar simple strategies can be applied to other cases, making clear that the approach turns out to be modular and flexible.

6 Results

In this section we analyze the performance obtained by our implementation of the *Splitter* module[3] (see table 1) and compare it with that of a standard Mpeg Video decoder. Before describing the experimental results, we observe that the decoder is just the first stage in the scheme of figure 2 and, typically, the least expensive one. If our implementation strategy proves to be quicker than the decoding stage, it would represent a significant improvement for application in real-time contexts.

To test the usability of our modules we processed a set of free MPEG2 Video test bitstreams [12], compressed using different compression factors. We then compared the

[2]More precisely we group in the first category all MPEG Video organization structures, side information and picture high energy DCT coefficients, and in the latter one all low energy DCT coefficients. Others partitioning strategies can be thought, like picture separation, etc.

[3]In the discussion, we focus only on this module since the *Degrader* and the *Merger* show a similar behaviour.

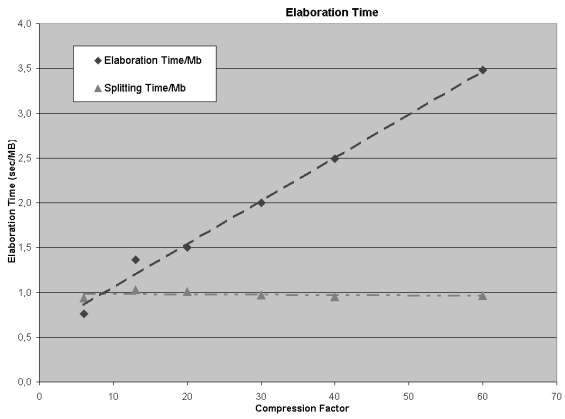

Figure 6. Processing Time.

processing time of the *Splitter* with decoding time. All tests have been made using an Intel Pentium III (500MHz) workstation with 256MB of main memory, running Windows 2000. The results using a file I/O interface are shown in figure 6, where the elaboration time per data unit (s/MB) is plotted against the compression factor.

We can observe that, at low compression factors, decoding performs slightly better than splitting, even though the *Splitter* overhead is negligible. At higher compression factors, the splitting time remains constant while decoding requires a linearly growing time. In the range of compression factors interesting for practical applications, our scheme performs much better than the decode-elaborate-decode scheme, since it limits decoding at a minimum. The low overhead observed at little compression factors, when the advantage of our scheme is lost, proves that our architecture and implementation strategy do not introduce a significant penalty.

7 Conclusions

In this work, we presented an abstract model for the structured data processing problem and derived a strategy to efficiently implement it using object-oriented mechanisms.

The key feature of the model is the encapsulation into a virtual parsing class of the structure information management. On the one hand, this choice isolates the problem to define the *operational-line*, i.e. to establish the trade-off between efficiency and flexibility of the implemented module. On the other hand, the virtual parsing class keeps separate that part of the elaboration depending on specific compressed formats, which can be the most difficult part to implement in practice. In this respect the availability of automatic generation tools for the abstract parsing class can be very helpful (see [11]).

Our future efforts will be focused on porting the implemented modules over an active network system ([10]), on creating a set of new modules based on the MPEG4 standard, on investigating the suitability of our system for the MPEG2/MPEG4 real-time transcoding.

References

[1] E. Amir, S. McCanne, and H. Zhang, “An Application-level Video Gateway”, *Proc. of the Third ACM International Conference on multimedia*, November 1995, San Francisco, CA, pp. 511-522.

[2] J. I. Khan, and S. S. Yang, “Architecture Overview of Medianet Multiprocessor Transcoder”, *Technical Report 2000-08-01*, Internetworking and Media Communications Laboratories Department of Math. & Computer Science, Kent State University

[3] J. Youn, M.-T. Sun, and C.-W. Lin, “A Study on Video Transcoder Architecture”, *IEEE Int. Symp. Consumer Electronics*, Oct. 1998, Taipei, Taiwan.

[4] J. Youn, M.-T. Sun, and C.-W. Lin “Motion Estimation for High-Performance Transcoders”, *IEEE Trans. Consumer Electronics*, vol. 44, pp. 649-658, Aug. 1998.

[5] J. Youn, M.-T. Sun, and C.-W. Lin, “Motion Vector Refinement for High Performance Transcoding”, *IEEE Trans. Multimedia*, vol. 1, pp.30-40, March 1999.

[6] J. Youn, J. Xin, M.-T. Sun, and Y.-Q. Zhang, “Video Transcoding for Multiple Clients”, *Proc. SPIE: Visual Communication and Image Processing*, Sydney, Australia, 2000.

[7] H. Sun, W. Knok and J. W. Zdepski, “Architecture for MPEG Compressed Bitstream Scaling”, *IEEE Trans. Circuits Syst. Video Technol*, vol. 6, No. 2, April 1996.

[8] Brian C. Smith, Lowrence A. Rowe, “Compressed Domain Processing of JPEG-Encoded Images”, *Real-Time Imaging*, volume 1, number 2, July, 1996, pp. 3-17.

[9] Soam Acharya, Brian Smith, “Compressed Domain transcoding of MPEG”, *Internationa Conference on Multimedia Computing and Systems*, 1998.

[10] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Philips, and J. Kann, “Introduction to the ASP Execution Environment”: www.isi.edu/active-signal/ARP/index.html.

[11] Flavor Project: http://flavor.sourceforge.net.

[12] MPEG2 Video test bitstreams at Tektronix: ftp://ftp.tek.com/tv/test/streams/Element/index.html.