Model-Based Decoder Specifications for the Long-Term Preservation of Video Content

Christian Schenk[**]

Computer Science Department
Universität der Bundeswehr München
85577 Neubiberg, Germany

Abstract. The long-term preservation of digital data implies two aspects that are substantial: The data's binary representation (i.e., the bit sequence) must be kept restorable in its original form, and the capability to decode (or interpret) the binary representation has to be retained. The former requires failure-resistant hardware, and, in case they have to be replaced, the existence of procedures that ensure copying without loss of information. The latter is a question of how the availability of compatible decoding software can be guaranteed. A general-purpose approach is to use standard formats that have been designed for or have proven to be well suited for long-term preservation. In case of digital video content, however, such a format does not exist. Common video formats are quite complex, and the conversion of digital video content without risking any loss of authenticity or information is usually not possible. As we assume that this circumstance will not change in the near future, we propose *decoder specifications* as a means to describe the decoding process using different abstraction mechanisms. Being human-comprehensible and machine-processable, such a specification serves as a template, which supports a potential developer to implement a decoder prototype for an arbitrary architecture. This way, it is ensured that most commonly used formats can also be decoded on future architectures.

Keywords: long-term preservation; video decoding; models; model transformations; H.264

1 Introduction

Documents, in general, are used to provide, publish and preserve any kind of information. While, in most cases, such documents are relevant for a limited time only, there are also documents that are important for future use (e.g., historical texts, photographs or videos). The issue of archiving such documents in a way that future generations can access them is addressed within the domain of long-term preservation.

Because of the digital revolution, today, we mostly work with digital documents, which are either digital-born (i.e., they have digitally been created) or

[**] Doctoral supervisor: Uwe M. Borghoff. The author is in his third year.

the result of a digitization step (e.g., using a scanner or a digital camera). Thus, it is worth thinking about the long-term preservation of digital documents [1]: As digital content is always stored as a sequence of bits (i.e., ones and zeros), which is likely to be transferable from one medium onto another one, it will always be possible to copy a digital document without any loss of information and to store it redundantly (even if, one day, a future architecture might use another representation for digital content). Consequently, every digital document can theoretically be preserved for an indefinite period of time.

Preserving, however, only makes sense if it is guaranteed that the digital content can also be restored in the future. As digital documents always need appropriate hard- and software when being restored, we have to focus on the question of how the availability of compatible software (called decoders in the following) can be ensured even for future architectures. As applications generally are not portable between different architectures, it can be assumed that the re-development of decoders will mostly be necessary when an architecture is replaced by another one. Consequently, whenever digital content is preserved, it must be ensured that the capability to develop such a decoder can be retained.

Standard formats, in general, are used to unify the serialization of digital content of a specific type (e.g., text, audio or video). Specifications that define such formats enable potential developers to implement a corresponding decoder. Therefore, standard formats are principally an appropriate means to ensure that preserved content can be restored on future architectures. However, while there are standard formats that have particularly been designed or (at least) have proven to be well suited for long-term preservation (such as PDF/A for textual documents or JPEG 2000 for images [5]), many widely used standard formats are appropriate for everyday usage but not in long-term preservation scenarios. Proprietary formats, for instance, are often not publicly available; other formats provide too much features that are useful in scenarios but impede the development of compatible decoders.

Migration, i.e., the transformation of digital content from one format into another one, is an approach that makes it possible to convert data into a format that is suitable for long-term preservation. This way, a Word document, for instance, could be migrated into the PDF/A format. Furthermore, migration can also be used whenever a new format is designed that is better suited for long-term preservation than the original one or whenever one format becomes outdated (and thus will no longer be supported). In both scenarios, migration is a (supporting) measure for the preservation of digital data. Though, as two different formats are usually not completely compatible to each other, every migration involves a loss of information or at least authenticity. Hence, in a preservation scenario, it is important to avoid migration steps wherever possible.

1.1 Preserving Digital Video Content

Digital video content usually is stored using a modern video serialization standard (such as H.264 [3]), which allows for high compression, high quality and efficient en- and decoding. Efficiency mostly is accompanied by system-dependence, and that is why modern serialization formats are commonly tailored to current architectures. Furthermore, they provide additional features to allow for an effective usage in different scenarios or for different purposes. Both aspects impede the development of compatible decoders, especially in cases where the target architecture is likely to be different. Hence, current serialization formats are ill-suited for the preservation of digital video content, but because there is no other suitable approach (to our knowledge), currently there is no alternative but to use (one of) these formats. In addition to that, there is another issue: modern video serialization formats use lossy compression techniques, i.e., the encoding of digital video content usually involves an irreversible loss of information. Thus, the migration of digital video content, which enforces recoding, should be avoided.

The problem of preserving existing digital video content can be summarized as follows: Currently, users who want to preserve a video need to store it in its original format (instead of a format that is at least well-suited for long-term preservation) and they have to rely on the corresponding format specification being explicit enough so that future developers can implement compatible decoders. Therefore, we are considering the following questions in our research:

1. How can existing digital video content be preserved so that its decoding on future architectures can be ensured without risking loss of information?
2. What can be done today to simplify the implementation of suitable video decoders on future architectures?

Our general idea is to define *decoder specifications*, which enable potential developers to implement a decoder on an arbitrary architecture [8], whereby different means of abstraction ensure such a specification to be human-comprehensible and also machine-processable.

In our current work, we propose a model-driven approach and want to evaluate if common MDE technologies (i.e., metamodels, model transformations, etc.) may serve as a basis to abstract the video decoding process for common video standard formats so that decoding capabilities can be retained in case an architecture has to be replaced. In this paper, we give an overview of this approach.

The remaining paper is structured as follows: In Section 2, we give an overview of related work and, in Section 3, we give a short introduction to video decoding. In Section 4, we explain the overall idea of our approach before we give an overview of the current implementation status in Section 5. Finally, in Section 6, we summarize the paper and give an outlook.

2 Related Work

We already have named two approaches that can be used for preserving digital content, namely *standard formats* and *migration*, and we explained why they are not suitable for digital video content. Migration usually implies recoding and thus results in loss of information. As a consequence, it would also be impossible to use one of the existing standard formats (such as H.264) to store arbitrary videos as it would always imply a migration step. Another approach, which focuses on the decoder's portability, is *emulation* [1]. In simple words, the architecture the decoder has been designed for is emulated on a (future) architecture in form of a virtual machine, which permits to use the original decoder implementation on a new architecture. This approach, however, only makes sense if an emulator for a complete system is really worth being implemented, for the development of a decoder normally is supposed to be easier. We briefly introduce some approaches that address these issues:

Image standard formats for the preservation of video data: Technically, a video can be regarded as a sequence of images (or rather frames). JPEG 2000 has already proven to be well-suited for the preservation of image files [5], and in addition to that, Motion JPEG 2000 [2], an extension of the JPEG 2000 standard, proposes to use the image coding procedure for every video frame. Hence, a combination could serve as a basis for archiving digital video content.

In [10], an XML-based language is proposed to store digital video content. Two variants are distinguished: first, video data are completely transformed into primitive XML; second, the video frames are converted into image files that are suitable for long-term preservation (such as JPEG 2000) and then referred to within the XML representation.

Contrary to our approach, the two approaches involve recoding, which generally results in loss of information. Furthermore, they only use still-image compression and, thus, compression results tend to be worse because similarities among different frames are not exploited.

Universal virtual computer: The decoder specifications, we propose, serve as templates for the implementation of concrete decoders for different architectures. Principally, they are a means to overcome the fact that ordinary software is not portable. The universal virtual computer (UVC [6]) addresses the same issue. It is a hypothetical computer that is supposed to be implementable in form of a virtual machine on arbitrary systems (i.e., it can be emulated). Originally, it has been designed as a general-purpose approach for the preservation of digital content. The basic idea is to write decoders in form of a UVC program. As such a program is portable between different UVC implementations, restoring preserved content (on a future architecture) can be limited to the development of a suitable UVC implementation.

The main difference between both approaches is the level of abstraction: While our decoder specifications are human-comprehensible and machine-processable, an executable UVC program consists of UVC machine code, which only supports basic instructions, also excluding floating point calculations. Therefore, a UVC program will usually be rather complex, difficult to maintain or to extend. Having this in mind, the development of a UVC program for a modern and complex video serialization standard seems to be unrealistic. However, a decoder specification principally could also serve as a template for developing a UVC program.

3 Digital Video Decoding

Before we describe our approach, we briefly provide an introduction to the decoding process of digital video content.

Every video can be regarded as a sequence of *frames*, which, when presented consecutively for a fixed period of time, create the feeling of movement. Regardless of how the video has actually been serialized, in every scenario, it is the decoder's task to restore the sequence of frames for further processing. Therefore, serialization standards are used that specify how the binary representation has to be interpreted during the decoding process. The H.264 video standard [3], which is used in combination with Blue-Ray disks, for video streaming scenarios and for locally stored video files, is one of these standards. As it is widely used, we use it for the following explanations, but we assume that most of the principles also hold for other standards that use similar or the same techniques.

An H.264 encoded video consists of *samples*, which are grouped into *chunks*. Principally, every sample contains all the information needed to reconstruct one frame. Every sample is partitioned into *macroblocks*, which provide the actual pixel data, and which each represent a region of 16×16 pixels within the frame. This structure is encoded within an H.264 serialization. The corresponding standard uses several methods that convert video data into representations that make further compression techniques effective. One of these methods (called *delta compression*) follows the principle of only storing differences between frames, macroblocks or even arbitrary regions of different frames. Most of these methods have in common that they are based on mathematical operations. These operations result in data representations usually containing values that tend to be small and occur quite frequently.

The H.264 standard combines different compression techniques, which are either lossless or lossy. Lossy compression algorithms play an essential role as they promise higher compression rates. Principally, these algorithms are used to detect "similar" values and transform them into one single code. Hence, the resulting representation contains more identical values than the original one and thus serves as a perfect input for lossless compression algorithms, which can handle repetitions effectively.

In summary, a common H.264 decoder first has to reverse the lossless compression in order to create an intermediate representation consisting of lossy compressed (as well as uncompressed) data. Then, it has to resolve dependencies and to reverse the lossy compression before actually being able to restore the pixel data. As a final step, the decoder has to ensure that the decompressed frames are put into the correct order that is specified by given time stamps.

4 Proposed Solution

Assuming that there will not be any appropriate standard format in the near future, we are considering the question of how a video, serialized in a common standard format, should be preserved if it is to be restored on future architectures. Instead of concentrating on the actual serialization, we pursue the goal of simplifying the development process for decoders by combining different means of abstraction to define human-comprehensible and machine-processable *decoder specifications.* Ideally, such a specification enables a developer, regardless of being familiar with the video standard or not, to implement a functional decoder prototype for any arbitrary architecture. Such a prototype, in contrast to a common decoder, only has to be able to restore the digital content, but it needs not to be efficient. Neglecting the efficiency aspect may have a positive impact as it is assumed to simplify the development process, which is actually essential. Nevertheless, such a decoder prototype (regardless of how efficient it is) is still applicable for different purposes:

1. It can be used to completely decode video content for further processing. In particular, the decoded content can also be recoded using an up-to-date format that is well supported on the target architecture.
2. It can be used for cases where efficiency is less important, such as for the extraction of single frames.

As a final remark, it has to be stated that the first scenario is not a normal migration step as its result is only used as an intermediate representation; it is not used for long-term preservation.

As a conclusion, we expect the decoder specification to be an appropriate means to retain decoding capabilities for existing video content. Such a decoder prototype can even serve as a basis for the development of an optimized and more efficient version or can serve as a reference implementation for an architecture-specific decoder.

For the following explanations, we assume that the video content has been encoded using the H.264 serialization format, but as said before, other formats usually follow similar or rather the same encoding principles and thus are supposed to be processable in a similar manner.

4.1 Model-based Decoder Specifications

In Section 3, we have described that general video content can simply be regarded as a sequence of frames. In the same section, we have also explained that an H.264 encoded video follows a hierarchical structure. Therefore, these two abstractions constitute the basis for specifying the in- and output of a suitable decoder. A decoder specification, thus, simply has to describe how one abstraction has to be transformed into the other. Assuming that a high level of abstraction guarantees the specification to be architecture-independent and human-comprehensible, we propose a model-based approach: in- and output is described using metamodels (as illustrated in Fig. 1 and Fig. 2).

We further have divided the complete decoding process into several steps, each transforming one or more data representations into another one, whereby all these intermediate representations are formalized by metamodels, which suggests that each step can be regarded as one model transformation. Indeed, we propose to use model transformations as a means of abstraction for several processing

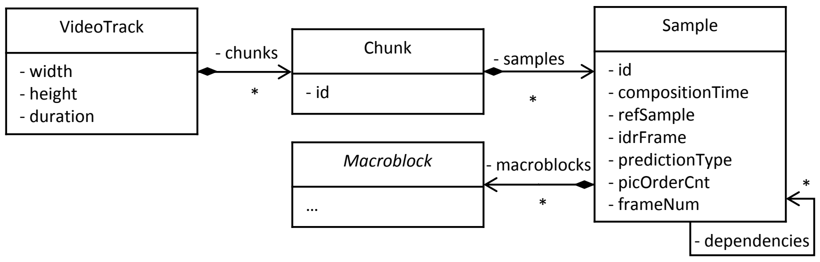

Fig. 1. H.264 data metamodel - due to clarity reasons, details of the abstract class `Macroblock` and all its subclasses have been omitted

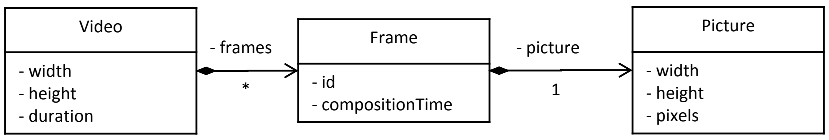

Fig. 2. Frame sequence metamodel

steps, but, as our main principle is to use abstractions that best fit a specific purpose, we also use other abstraction mechanisms if they are supposed to be more suitable. As video decoding usually involves mathematical calculations, we use an abstraction mechanism being well-suited for specifying mathematical expressions and operations.

We have grouped all the decoding steps into five phases (illustrated in Fig, 3), whereby the *modeling* and the *unmodeling phase* serve as pre- and postprocessing steps that convert a video into an H.264 model and, vice versa, a frame sequence model into a sequence of frames. These phases highly depend on the actual serialization format, the target architecture and the concrete scenario; therefore, they are not in the scope of the decoder specification.

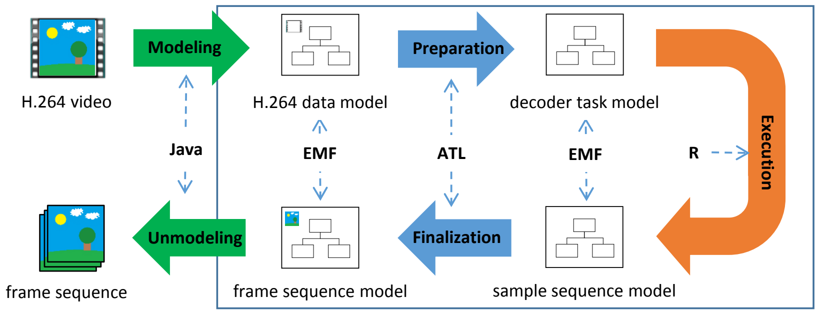

Fig. 3. Model-driven decoding process for an H.264 video

The *preparation phase* transforms the original H.264 model into an intermediate representation that constitutes the basis for performing the mathematical operations within the *execution phase*, which are necessary to restore the pixel data. Thereafter, the *finalization phase* creates the output model containing the uncompressed frame data. These three phases are the essential steps of a decoder specification. In contrast to the execution phase, which is defined using mathematical abstraction mechanisms, the preparation and the finalization phase are specified as a series of model transformations.

The main advantage of our specifications over common informal ones is the fact that they are designed to be human-comprehensible **and** machine-processable. This implies that they constitute the basis for automatic code generation and/or direct execution. Besides the obvious effect that a developer will not have to implement the complete decoder from scratch, the proposed specifications are supposed to be more unambiguous than common informal format specifications. This also implies that all the intermediate model representations, resulting from the particular decoding steps, should always be the same regardless of the concrete implementation. As a consequence, it is possible to generate test models that, if provided in combination with the decoder specification, permit developers to test the implementation of every step by comparing the resulting model with the corresponding test model. This way, developers can identify and correct implementation errors; consequently, the development process can further be simplified.

4.2 Preservation Process

As stated, a decoder specification only describes the steps that are needed to transform an H.264 model representation of an encoded video into a frame sequence model. Therefore, we propose to store the H.264 model in a serialization format that is well-suited for that purpose and that is likely to be restorable in the future (such as XML). By using a general-purpose compression algorithm, the file size can further be reduced without risking loss of information. This way, restoring the model representation (i.e., implementing the modelling phase) is likely to be easier than it would be if the original file were used.

In summary, if a video is to be preserved, its model representation have to be converted into the serialization format and stored in combination with the decoder specification and additional test data (as described before). When it is restored one day, developers first have to restore the model representation; afterwards, they can use the decoder specification to develop a suitable decoder.

5 Current Status and Future Plans

The general applicability of the frame sequence model is a requirement for the definition of decoder specifications that allow for the implementation of concrete decoders for different scenarios. Therefore, we have implemented a library demonstrating that the frame sequence model is applicable to reference and access video content in an interoperable way (as originally discussed in [8]).

Above all, based on the official specification [3], we have implemented essential parts of the decoding process for H.264 encoded videos using Java. While the decoding of dependent frames is not completely supported yet, independent frames and other information can randomly be accessed and extracted. Having implemented it from scratch (without using external libraries) with a focus on the abstraction (rather than efficiency), it serves as a basis for our current work, i.e., the specification of suitable decoder specifications.

Currently, we are working on the decoder specification's design for H.264 encoded videos. We have already implemented the preparation and the finalization phase using EMF metamodels [9] as well as ATL transformations [4], and we have defined a set of R scripts [7] that perform all the mathematical operations needed within the *execution phase* to restore the pixel data of independent frames. Furthermore, we have defined a DSL to write a control program that allows us to specify, which abstraction mechanisms (i.e., ATL transformation or R script) have to be used to transform one data representation into another one.

While the essential elements of the decoder specification, namely those of the preparation, the execution as well as the finalization phase, already exist and can also be executed, up to now, the transition between the three phases, i.e., its specification, is still hard-coded. In a next step, we want to replace the hard-coded creation of R-compatible data and the execution of R scripts (using the open-source library renjin[1]) with suitable abstractions such as model-to-text transformations. This will be the last step before we address the issue of decoding dependent frames.

The size of the intermediate model representations was a challenge we had to tackle before we were able to actually execute parts of the decoder specification (using MDE technologies): Because of the memory video data tend to require, using metamodels for their formalization results in large models. A test video of about 2 hours with a resolution of 1280×720 pixels, for example, resulted in an H.264 model that, if represented by a graph, consists of about 14.5 billion nodes and 15.7 billion edges. As the complete video decoding process allows for effective partitioning, in our current solution, we use model representations only containing those contents that are actually needed for every phase. For performing the preparation phase of the aforementioned example model, we were able to use a reduced graph representation, which contains "only" about 327,000 nodes and 646,000 edges.

As a next step, we plan to test our approach by asking developers to implement a program that can convert H.264 models into frame sequence models. For that purpose, the developers will be given an H.264 decoder specification, an H.264 model of a video and additionally all the intermediate model representations resulting from the specified decoding steps as test data. We will assume our approach to be a full success if the resulting decoder prototypes can actually be used to transform H.264 models into frame sequence models. However,

[1] Project URL: www.renjin.org

we also want to find out whether the chosen abstraction mechanisms are well suited for specifying the video decoding process or whether other mechanisms and technologies should be preferred.

Thereafter, we plan to test our concept for other formats in order to evaluate its general applicability. In this context, we also want to examine if parts of the H.264 decoder specification can be reused for other specifications.

6 Conclusion and Outlook

As stated, we propose decoder specifications that help potential developers implement functional decoder prototypes, even if they are unfamiliar with a video standard. Our work has been motivated by the lack of existing approaches that are supposed to be suitable for the long-term preservation of digital video content. Our approach directly addresses this issue and thus may also serve as a template for related problems. As decoders are ordinary software, the approach might be usable for other complex data, e.g., games.

A decoder specification unifies different abstraction mechanisms that ensure such a specification to be human-comprehensible and machine-processable. The means, we have chosen so far, allow for automatic code generation on the one hand, and they permit potential developers to perform such specifications in an early state of the development process on the other hand.

References

1. Borghoff, U.M., Rödig, P., Scheffczyk, J., Schmitz, L.: Long-Term Preservation of Digital Documents, Principles and Practices. Springer-Verlag Berlin Heidelberg (2006)
2. ISO/IEC: International Standard ISO/IEC 15444-3: JPEG 2000 Image Coding System - Part 3: Motion JPEG 2000. International Standard Organization (2002)
3. ITU-T: Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services. International Telecommunication Unit (2013)
4. Jouault, F., Allilaire, F., Bzivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(12), 31–39 (2008)
5. van der Knijff, J.: JPEG 2000 for Long-term Preservation: JP2 as a Preservation Format. D-Lib Magazine 17(5/6) (2011)
6. Lorie, R.A., van Diessen, R.J.: UVC: A Universal Computer for Long-Term Preservation of Digital Information. IBM Research Division (2005)
7. Ross Ihaka, R.G.: R: A Language for Data Analysis and Graphics. Journal of Computational and Graphical Statistics 5(3), 299–314 (1996)
8. Schenk, C., Maier, S., Borghoff, U.M.: A Model-based Approach for Architecture-independent Video Decoding. In: 2015 International Conference on Collaboration Technologies and Systems. pp. 407–414 (2015)
9. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edn. (2009)
10. Uherek, A., Maier, S., Borghoff, U.M.: An Approach for Long-term Preservation of Digital Videos based on the Extensible MPEG-4 Textual Format. In: 2014 International Conference on Collaboration Technologies and Systems. pp. 324–329 (2014)