SECURE GLOBAL PRODUCTION – USING THE JT-NM REFERENCE ARCHITECTURE TO BUILD CLOUD/HYBRID WORKFLOWS

Dr James Westland Cain

Grass Valley, UK

ABSTRACT

This paper presents a detailed technical experience report of developing a hybrid On-Premise / Cloud offering for Live and Production workflows.

All media has strong Identity (every frame has a name). All media is accessed using the Flows and Grains model as described in the JT-NM Reference Architecture.

Consequently, it becomes trivial to build hybrid workflows combining On Premise and Cloud storage for essence.

Client applications can choose the quality of any essence on a frame-by-frame basis. This empowers all edge devices; offering global scalability and truly novel workflows.

Client applications are also able to ask for arbitrary ranges of essence. Asking for a single GOP from the middle of a two-hour record – from the other side of the planet – is now possible with this model; saving time, eliminating waste.

With this power comes great responsibility. Therefore, every Grain access is secured with state-of-the-art authentication / authorization checks and full auditing.

INTRODUCTION

According to Butler Lampson, a technical fellow at Microsoft Research, the fundamental theorem of software engineering is *All problems in computer science can be solved by another level of indirection* Spinellis (1).

The media production industry has inherited so much from computer science; The PC, Networking, The Internet, Files and Filesystems, and more recently Cloud Infrastructure. The JT-NM Reference Architecture (JT-NM RA) (2) is an attempt to encourage us to also add a level of indirection into our architectures.

This paper will set out why this is necessary, and what benefits this brings.

THE PROBLEM

File based media production has revolutionised the collaborative generation of media programming. However, the explicit use of files and filesystems has exposed a swathe of problems that then need to be solved by expensive and oftentimes brittle Media Asset Management systems.

When we use a file in a computer system, we ask the operating system (OS) to open the contents of that file by supplying a filepath to the OS APIs. These filepaths are translated into byte range accesses by the OS, such that correctly formatted media can be parsed by application software coded to expect standardised file formats.

Think about MXF as defined by SMPTE 377M (3) and later standards. The KLV substrate exposes a heady mix of metadata in standardised dictionaries. Both the index and the essence are co-located in the file. The OP1a Operational Pattern defined by SMPTE 378M (4) that constrains most MXF based workflows enforces that only one form of any essence for each track is contained within the file. Further the only way to reference that media is by opening the file using that filepath.

The problems filepaths introduce are many:

- Access to the file requires that the software is physically close to the media storage (location).
- The software gets hooked on the format that the essence is presented in (form).
- There is no ability to add any processing steps between the contents of the file and the application making the byte range accesses (process).
- The final problem, and the most insidious, is that the filepath is in effect a proxy for the identity of the media (identity). For example, the filepath will be added to any project files whenever the media is used in an editor timeline. Therefore, the filepath – which ingrains the form and location of the media – ties any project file references to exactly that – the current form and location of the media.

There are many modern use cases that this approach does not easily support, such as access over WANs using HTTP(S) (and by implication Object Storage access), or the use of a proxy version of the media. Whenever the media is copied or reformatted, we in effect change the identity of the media, as we assign it a new filepath.

Our industry has come up with clever mitigations that try to reduce the impact of these issues, such as client applications that have been adapted to support multiple filepaths concurrently, enabling hybrid proxy / quality workflows, or clever WAN optimised file transfer technologies, so that distant files can quickly be copied locally allowing for low latency access.

In both these examples we have increased complexity; for the Media Asset Management system, as multiple file copies need to be tracked, for the video editor as project files need to be conformed each time anything gets copied.

Our industry has also come up with many expensive workflow practices to counter these issues, such as copying all media between numerous sites in a customer's facilities just in case it is needed. Of course, this policy incurs exponential cost as sites increase. As our industry undergoes increasing merger and acquisition activity, these solutions just don't scale.

THE SOLUTION

In other parts of computer science similar problems to these have been elegantly solved. The solution, which normally entails another level of indirection (otherwise known as information hiding Parnas (5) or encapsulation), involves giving things names.

Names need to be stable irrespective of form or location. An example, by analogy is my mobile phone. My phone number (which is a proxy for my identity) is stable. I can be called using that number wherever I happen to be. My location is not exposed by my mobile, and my identity remains stable as I move about.

Identity is not coupled to the form or location of the media it represents. Take a picture of me. It is still the same essence when formatted as a jpeg or a png. I can re-size the picture, make a thumbnail, email it to a friend, or even print it as a photo. It is still a picture of me. Its identity is stable, irrespective of form, location or processing.

This property of stable identities is key to the reference architecture described by the JT-NM RA.

SourceGroups, Sources, Flows and Grains

One of the key abstractions in The Reference Architecture is the relationship between SourceGroups, Sources, Flows and Grains.

All these types have a strong Identity (or Stable Names).

SourceGroups contain contemporaneous Sources from a recording. Sources provide access to one or more Flows of Essence. Flows offer essence at a particular rendition (form) by enabling access to one or more Grains of Essence. Each Grain contains a contiguous range of essence in said rendition.

- Grains are units of essence – frames or GOPs of video, samples of audio, even ANC payloads.
- Flows are streams of grains that have the same format (or rendition).
- Flows also map Time to Bytes. Thus, they can be seen as an index abstraction.
- Sources are tracks of a single type of essence, such as video audio or data. By offering multiple Flows, Sources allow access to the same media at different qualities.
- SourceGroups are a recording. The set of Sources recorded at the same time.

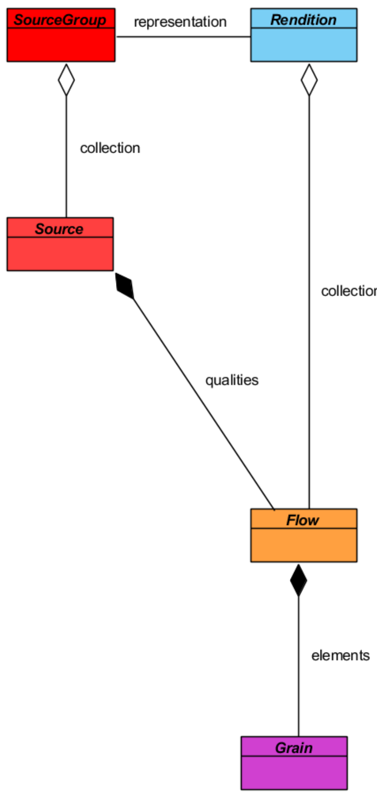

Figure 1 – UML for SourceGroups, Sources, Flows and Grains

Interfaces are Contracts

These set of abstractions can easily model the parts in an MXF OP1a file. A file is a SourceGroup, each track is a Source. Each Source only offers one Flow due to OP1a constraints. The Flows represent the indexes in the MXF, giving access to the byte ranges in the file that are Grains in this new abstraction.

These abstractions are also a suitable base to build a web server that can offer grains as a RESTful API, a technique first described by Fielding (6). So, what is a file can also be streamed. You cannot consume an MXF file in a browser. However, a browser will happily consume RESTful APIs that offer the JT-NM RA abstractions as URLs.

These abstractions form a contract. The interfaces are all you may know about the nature of the source you are trying to access. The fact the media is currently stored in an MXF file is not known to you. The fact that more than one copy may exist, that some parts may have been transcoded into proxy, that some locations are Cloud stores and that some proxy is only created on the fly when asked for are all things that must be hidden from all client applications if we are to hide the form and location of our media.

By this interface, we can break the hegemony of the filepath that has come to dominate the collaborative workflows in our industry; exposing the filepath has disabled our client applications from taking advantage of all the WAN access and Cloud infrastructure that modern Computer Science has built.

TIME

The Flow abstraction represents an index, a mapping between time and byte ranges. What units does time need to be counted in? In a departure from the JT-NM RA, the implementation chosen was frames. Counting frames is sufficient for editor timeline access.

Most forms of media are fine with frame-based counting, video obviously is, embedded data such as captions also are. The audio we support is sampled at 48KHz, so 1001 based media can be grouped into 5 frame multiples to get whole number maths (8008 audio 48KHz samples is exactly 5 frames in NTSC). AAC encoded audio can also be modelled using simple internal offsets within grains.

This simplification – using frame-based access to grains – is not the only way time could be modelled, as Presentation Time Stamps, or 90KHz based counting would have been another very suitable unit of access. However, having a simple frame-based count to access frame ranges has made most of the implementation simpler than it might have been if we had exposed all the complex forms of time representation our industry has invented.

WEB FIRST

In the world advocated by the JT-NM RA, all Sources of media must offer the Flows and Grains access model. This implies that all sources of media can be accessed over HTTP(S). This has many benefits.

The MPEG-DASH (7) standard offers an abstraction such that browsers can access the same media at different bitrates. The different streams of media are fungible – one stream's decoded frames can be interchanged with another stream's frame accurately – you get exactly the same pictures, but at a different perceptual quality. This enables the browser to choose what resources to consume when its displaying media to a viewer. This technique is commonly known as Adaptive Bitrate Streaming (ABS) (8).

Of interest is the relationship between the track being played, and the different qualities of the same media being made available. This kind of relationship is what is being expressed when one considers how a Source can offer many Flows of the same essence – just at different renditions (by just varying compression bitrates and picture sizes).

However, the JT-NM RA Source to Flow relationship can offer far more than just different sizes and compressions of h.264 (9). There is no constraint on the Flows containing uncompressed media, a series of JPEG stills, or media with an HDR colour space. There is also no need to align the payloads so that they have I-Frames that make them interchangeable without running multiple codecs. In other words, nearly all media forms fit within the Flows and Grain model.

What the Source to Flows relationship does is enable the client to choose the best form of the media it requires given the available choices, rather than constrain what it can do by the limited choices of compression that it may get given when inspecting the contents of a file.

In the case of a browser playing video over the internet, ABS has proven its worth. In the JT-NM RA model this enables many more workflows beyond what a browser can consume. For example imagine a render engine that chooses to take its time to acquire the best quality pictures available, or an archive workflow that needs a the very highest quality copy of a recording, but can choose to trickle the media when networks are not busy at non-peak times, or a playout workflow that needs to make real-time, the playout tool can use the same algorithm that the browser uses when playing to achieve the best quality possible, given the current network constraints.

URLS

RESTful URLs allow the client to discover different options by making simple GET requests and then follow links supplied using the resources returned. Given a Source Identity, what information can a client garner from the URLs on offer? We can construct a URL to discover information about that Source. For example, what Flows does it offer? A simple GET request to https://<hostname>/01F3Z436KSWGCS12301R09305F/flows,

returns JavaScript Object Notation (JSON) containing one or more Flow descriptions as follows:

```
{
   "flow":"https://<hostname>/01F3Z436KSWGCS12301R09305F/hq",
   "descriptor":{
      "type":"video",
      "compression":"h.264",
      "size":{
         "x":1920,
         "y":1080
      },
      "rate":{
         "nom":50,
         "denom":1
      },
      "bitrate":2500
   }
}  [This snippet has been shortened for brevity's sake].
```

Of note is the embedded URL that can be followed to access the flow. The flow has been named “hq”, though this has no technical meaning, we just need each flow to be uniquely identified. There is enough information in the descriptor for the client to decide which flow to open.

On getting https://<hostname>/01F3Z436KSWGCS12301R09305F/hq we are returned more JSON:

```
{
   "grains":"https://<hostname>/01F3Z436KSWGCS12301R09305F/hq/{offset}",
   "ranges":[
      {
         "offset":0,
         "duration":1500
      }
   ]
}
```

Here the grain URL indicates it is a template RFC6570 (10), that requires the insertion of a number (the meaning of {offset}). Suitable numbers are indicated by the ranges structure.

If we then create a URL asking for the grain at offset 10, we get:
https://<hostname>/01F3Z436KSWGCS12301R09305F/hq/10.

The response is a binary payload, but of interest is the HTTP headers the response carries:

```
"Grain-Offset": 0
"Grain-Duration": 15
```

The payload that the grain has produced carries 15 frames of video. Even though we asked for frame 10, we got a grain starting at offset zero. This is another form of encapsulation in action, as we only discover the way grains are packed into payloads at runtime.

Also, of note, but not shown, is that Sources and Flows can also adhere to the MPEG-DASH spec. This is great as browsers that are unaware of the JT-NM RA inspired URLs can just play flat video as if it were normal ABS media.

SECURITY

This model has made all frames in all formats available as URLs to anyone who cares to GET them. There are no physical limits to this model. Anyone with internet access can get at all media.

To counter this, we need to secure every single URL to only those who are authenticated and authorised to get access. Further we need an Audit trail of who did what when.

The AMWA BCP-003 (11) Security recommendations for NMOS APIs offer a compelling solution to these issues. Only support HTTPS (when run over TLS 1.2 or better) so that all traffic is encrypted and no man in the middle replay attacks are possible. Once all traffic is away from prying eyes, require that all URLs carry an OAuth2 OIDC JWT Bearer token RFC7523 (12).

A JSON Web Token (JWT) as defined in RFC7519 (13) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims are digitally signed by a cryptographically secure signature to ensure nothing has been tampered with. The clever thing about the signatures is that they are checkable without needing to contact the issuing authority each time.

A digital signature scheme typically consists of three algorithms:

- A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
- A signing algorithm that, given a message and a private key, produces a signature.
- A signature verifying algorithm that, given the message, public key and signature, either accepts or rejects the message's claim to authenticity.

The JWT then has the property that the authenticity of a signature generated from a fixed message and fixed private key can be independently verified by using the corresponding public key.

This means our web server endpoints do not need to contact a single Identity endpoint when checking each HTTPS GET request. They can issue a 401 Unauthorized quickly and independently.

Setting up an Identity scheme based upon this is beyond the scope of this paper, but commercial offerings are available from all the major Cloud vendors.

FILES BECOME STREAMS

When I make a query using google.com, I get search results, but I cannot tell where the computers were that serviced my requests. That is the property we need when accessing media. The JT-NM RA advocates this. All file access is via secure streaming style URLs that hide the form and location of the originating source. Files obviously are still crucial to all workflows, as are shared disk systems, but where they are stored has been hidden.

This allows much freedom, to re-locate the files from an on-premise recording, to a Cloud hosted object store acting as a long-term archive. The moving of this media does not affect the clients of these URLs at all.

There is a need to track where copies of the files are located, and to route requests for essence to the most appropriate copy. This can be achieved by a simple locations store that can keep an index of what copy, at what rendition, is stored where.

The client needs to know the key – the Identity – to cause the correct locations records to be queried. That is fine as the client is using Identity carrying URLs to make its access requests. We can then use a simple database query to find the most suitable form and location of media to read from.

There is another freedom born of this indirection. We can vary the location of the client. Given this post pandemic world we are living in, home and remote working has never been more topical. As long as the client has the correct permissions, they may be located anywhere there is sufficient bandwidth to service their requests.

ACCESSING STREAMS FROM EDITORS

We cannot overnight change all the application software we use. Also, we cannot yet enable browsers to deal with the plethora of compressions and formats that our industry has invented. So, there is a need to join installed applications into the stream-based ecosystem.

An approach that has proven very effective is to exploit the plugin APIs that almost all software that imports files offer. For example, Adobe Premiere offers a rich plugin environment to enable new file formats to be accessed by the editor.

Given the Premiere Plugin API (14), we can author a plugin that can use the URLs on offer to acquire grains of essence in the best form available for the current use case. The client here obviously needs to be authenticated, but this allows us to access streams as if they were files.

We can then use appropriate pre-cueing algorithms and local disk-based caching to allow the Premiere system to not know it is accessing media over HTTP. We can also vary the quality of what is fetched, so that real-time guarantees are honoured.

We can then offload the rendering to a render farm – that is cloud hosted, or certainly not constrained to be co-located.

Of note here is that project files do not need to be conformed as we move them – globally – as they contain no references to physical media. They are not constrained by material form or location. The fact we change media quality or relocate media sources is quite hidden from them.

Many other editing systems offer Plugin APIs, such as Avid Media Composer, or Grass Valley Edius. An obvious cost here is that this does require that each plugin module has to be developed and maintained. The benefit is that the software works irrespective of the form and location of the source, or the location of the editor, or the location of the playout / render target.

It has freed us from having to be in the buildings that host our media. It has also freed our media from having to be in those buildings!

We can therefore easily build “multi-location” timelines. Some media is local, some in S3 in AWS, some in Google Cloud Storage, some even being live ingested on the other side of the world.

OPTIMISATION

If an editor or any other client is co-located with some media storage, do we really need to access this via a web server over HTTPS? If the editor is browser based, then the answer is currently yes, as we want a uniform API from the browser developer's point of view.

This analysis changes with a plugin hosted within an installed application, such as Adobe Premiere. The plugin can choose to directly access the files using classic filepaths, with strict caveats. Whilst the filepath is used to access media bytes, the fact the plugin is doing this must not be exposed to the wider application. Filepaths must not be allowed to leak into the project files that Premiere creates, otherwise the project file becomes overtly coupled to the form and location of the media, voiding the identity indirection we have worked so hard to add.

Direct access not only saves the additional expense of running web servers in front of the storage, but also mitigates the one problem that this architecture does not inherently address: very low latency access. An example use case here is high speed scrub.

File and operating systems combined with application software have had decades to optimise the efficient access of the bytes stored within the file. With very high-speed Local Area Networks connected to well-designed scalable Shared File Systems, the application software can expect near wire speed throughput with incredibly low latencies. This expectation is truly hard to meet using a webserver in front of the storage.

The plugin's job is to translate identities and time offsets into grains in the form that the host of the plugin requires. How it does this is encapsulated, so it can access URLs or UNC filepaths just as easily.

This direct access idiom can also be used when accessing cloud storage hosted media files. For example, an MXF file stored in AWS S3 is already accessed over authenticated HTTPS – so there is no real need to add an additional web server as an intermediary in this case either (on the proviso that the client will need the correct AWS credentials for this direct access).

CONCLUSION

In the model just described, media can be ingested anywhere, media may be stored in the cloud or on premise. Staff can be anywhere there is a reasonable internet connection. Playout and Deliverables can be run on premise or in the Cloud.

All this works, scalably, globally and securely. By hiding the material form and the essence location, we have enabled truly global workflows. By removing the hegemony of the filepath, we have freed our media workflows from being constrained by the walls of a building. We have also removed the need for many of the things our industry has built to compensate for the over exposure of media form and location.

Editors no longer need to conform – as media location is tracked and made available without changing the media's identity. Having many copies of the same media no longer implies a huge cost of media management, as moving the media does not entail re-asserting that you have the correct media for that part of the timeline.

Editors no longer need to track many different filepaths to enable proxy / quality hybrid workflows, as this facility is built into the access methods that the plugins the editors host now use.

Instead of having many isolated islands of media production, with little ability to join them together into a cohesive workflow, we have made an archipelago of islands well connected by a sea of media. This media mesh, with full unfettered access to those with the correct credentials, is very empowering, and portends a sea change in our approach to media production and delivery.

The exciting thing is this is no longer just a vision; it is a reality.

It has been built and it is in production today.

FUTURE – STREAMS INTO FILES

The requirement to write a plugin to embed into every application that wishes to access the media using these techniques is quite a burden. Could there be a more generic way to enable 3rd party applications into the ecosystem?

All applications that deal with media by definition read files. So, a powerful technique would be to build a Virtual File System (VFS), that offered files that any application may open. The clever part of this approach is that the files are made to order using the streaming APIs this paper has presented.

One thing to bear in mind is that these appear to be real files, with everything that entails, including all the problems that filepaths engender as described at the beginning of the paper.

The mitigation is that the virtual files themselves can have their content adapted from the streaming source into a form that is suitable for local consumption. So, all issues (form, location and processing) are in effect hidden with this approach. The only access requirements are a logged in user (with their credentials limiting what material they may access) and the media identity that they require – in the form of a filepath. The format and the range of grains can be adjusted by altering the file name and extension that the client opens. For example:

```
\\<hostname>\<share>\01F3Z436KSWGCS12301R09305F\0-1500.mxf
\\<hostname>\<share>\01F3Z436KSWGCS12301R09305F\10.jpeg
```

Such a VFS is not yet in production, but prototypes have been built that demonstrate the proof of concept.

REFERENCES

1. Spinellis, D. 2007. "Another level of indirection". In Oram, Andy; Wilson, Greg (eds.). Beautiful Code: Leading Programmers Explain How They Think. Sebastopol, California: O'Reilly and Associates. pp. 279–291.

2. JT-NM RA, 2015. Joint Task Force on Networked Media (JT-NM), Phase 2 Report, Reference Architecture v1.0, 4 September, 2015. https://www.jt-nm.org/reference-architecture, retrieved 12-04-2021.

3. SMPTE 377M, 2011. "ST 377-1:2011 - SMPTE Standard - Material Exchange Format (MXF) — File Format Specification," in ST 377-1:2011, vol., no., pp.1-183, 7 June 2011, doi: 10.5594/SMPTE.ST377-1.2011.

4. SMPTE 378M, 2004. "ST 378:2004 - SMPTE Standard - For Television — Material Exchange Format (MXF) — Operational pattern 1A (Single Item, Single Package)," in ST 378:2004, vol., no., pp.1-16, 22 Sept. 2004, doi: 10.5594/SMPTE.ST378.2004.

5. Parnas D. L. 1972. "On the Criteria To Be Used in Decomposing Systems into Modules". Communications of the ACM. 15 (12): 1053–58. doi:10.1145/361598.361623.

6. Fielding, R. T. 2000. "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.

7. MPEG-DASH, 2019. ISO/IEC 23009-1:2019 Information technology — Dynamic adaptive streaming over HTTP (DASH).

8. ABS. https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming, retrieved 12-04-2021.

9. H.264, 2019. ITU-T H.264, International Standard ISO/IEC 14496-10 – MPEG-4 Part 10, Advanced Video Coding.

10. RFC6570, 2012. URI Template. https://tools.ietf.org/html/rfc6570.

11. AMWA BCP-003. https://specs.amwa.tv/bcp-003/, retrieved 12-04-2021.

12. RFC7523, 2015. JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants https://tools.ietf.org/html/rfc7523.

13. RFC7519, 2015. JSON Web Token (JWT). https://tools.ietf.org/html/rfc7519.

14. Premiere Plugin API, 2021. https://ppro-plugins.docsforadobe.dev/index.html, retrieved 12-04-2021.

www.grassvalley.com/patents

ACKNOWLEGMENTS

The author would like to thank Chuck Meyer, Grass Valley Technology Fellow, for feedback and guidance on early drafts of this paper.