SMPTE

MOTION IMAGING JOURNAL

Covering Emerging Technologies for the Global Media Community

Advancements in Information Technology Applied to Professional Media

By Brad Gilmer

Introdução:

O artigo desta edição discute os avanços em tecnologias da informação aplicados à indústria de mídia. Algumas das ideias abordadas incluem a aplicação de princípios de TI e conceitos de computação em nuvem para sistemas de mídia, mas, desde o meu ponto de vista, o principal ponto deste documento é quando discute a importância da interoperabilidade entre sistemas e a capacidade dos sistemas de mídia de serem adaptáveis e escaláveis para lidar com fluxos de trabalho em constante mudança. Isso sim é musica para qualquer ouvido que esteja atento ao mundo líquido que estamos enfrentando. Entender, cada vez mais isso, é imprescindível para todos os profissionais. Então se eu fosse você não deixava esta chance de atualização passar batida!!!

Tom Jones Moreira

Abstract

Media companies are under ever-increasing pressure to create and monetize content efficiently by delivering it to viewers wherever they are and on whatever device they wish to use. To meet this challenge, media companies need flexible, composable systems that may be quickly adapted to shifting workflows and a rapidly changing business environment. This article introduces fundamental concepts from the IT domain and describes a way of thinking about a media system that enables flexibility and unlocks a powerful world of composable, dynamic media systems and workflows.

Keywords

Composable, dynamic media systems, immutability, IP systems, user requirements, workflow

Introduction

Media companies are under ever-increasing pressure to create and monetize content efficiently.[1] This pressure is created, in large part, by consumers seeking to view content wherever they are and on whatever device they wish to use. These devices and platforms are evolving at a dizzying pace. To make things more complex, viewers seek experiences that range from traditional long-form content with minimal interaction to dynamically created content that relies on interactions with viewers. When it comes to producing, delivering, and monetizing content in this environment, the challenges are manifest.

To meet this challenge, media companies need dynamic, flexible, and composable systems that can be quickly adapted to shifting workflows and a rapidly changing business environment. A blending of best practices from the IT and computer science domains, with a deep understanding of the professional media industry, may provide a way to address the challenges.

Media companies are under ever-increasing pressure to create and monetize content efficiently. This pressure is created, in large part, by consumers seeking to view content wherever they are and on whatever device they wish to use.

Definitions

"Professional Media" encompasses show production, post-production, editing, audio production, program playout, streaming service origination, and other similar areas. "IT" refers to the field of information technology.[2] Computer science or "CS" refers to a branch of science that deals with the theory of computation or the design of computers.[3]

Business-Driven Problem Statements

The following statements have been captured from several media executives as they seek to develop the next-generation media facilities.

"I want to be able to swipe a credit card and turn up a service. When I am done, I want to pay for what I have used. No CapEx." Richard Friedel –Fox

"For EuroSport's next set of facilities, we want a cloud approach for physical kit, we want common access to all content, and we want distributed operations that deliver remote production for all." "We are building a private cloud for live production." Gordon Castle –Discovery/Eurosport

"I don't want to be 'Dr. No', always telling people why they can't do something. I want a dynamic infrastructure that allows the business to try out new things without breaking the bank." Brad Gilmer –Turner Broadcasting

"We are putting the final touches on our major networked media facility. Our ability to innovate will continue to grow as long as the industry adopts more IT-thinking in their applications." Felix Poulin – CBC/Radio–Canada

How can we deliver the vision reflected in the statements above? Is there a body of work we can study or a group of practices we can follow that will make it easier to deliver on this vision?

Way Forward to Realizing the User Visions

A way forward would be to apply IT best practices to our domain of professional media. We know that IT and computer science have been focused on delivering the "ilities" (agility, flexibility, scalability, composability, and so on) for years. How did they do it? What are the vital game-changing concepts that have made IT systems what they are today? What specific things do we need to focus on at the architectural core to build a new kind of media facility?

Steal Everything

The author has developed a list of things we can steal from the IT domain and apply to IT systems for media. These represent a list of best practices (sometimes called *patterns* in the IT world) and fundamental concepts that have allowed computer scientists and IT professionals to develop the modern, scalable, responsive, and well-designed IT systems, and that are in wide use today.

Of course, the IT industry has its failures, some of them more spectacular than others.[4] And there are definitely things we should never steal from the IT domain (these are called *antipatterns*). In a way, the media industry is in a great position. We have the foresight of hindsight. This means we can carefully study what has worked and what has not worked in the IT domain and adopt the successful patterns that best suit our industry.

You will note many references to the Joint Taskforce on Networked Media (JT-NM) Reference Architecture[5] (JT-NM RA). This document, jointly published by the European Broadcasting Union (EBU), SMPTE and the Video Services Forum (VSF), is relevant in any discussion of IT systems for media applications.

Resources and Immutable Objects

The IT industry faced a number of problems with regard to how they thought of "things" as the internet exploded in the late 1990s. In 2000, Roy Fielding introduced powerful concepts regarding resources in his groundbreaking dissertation titled, "Architectural Styles and the Design of Network-based Software Architectures."[6]

In Fielding's architecture, everything is considered a resource. Applying this to the media facility, a movie such as *Gone with the Wind* (GWTW) is a resource. A specific version of that piece of content (edited for time, with subtitles and descriptive video, etc.) is also a resource. But what about a production control room? What about a studio camera? A specific playback server? All of these are resources. What about people—an audio operator, engineer, or technical director? These are resources. Are virtualized instances of a transcoder in the cloud considered resources? Yes. *Everything* in your facility is considered a resource.

There is a lot of power in viewing everything as a resource, especially if you apply some of the other concepts in this article. But we need a reliable way to refer to all these resources. Identity frameworks and immutable object identifiers provide a solution to this problem.[7] The JT-NM RA Identity Framework consists of three key points summarized below.

- Resources shall be identified using IETF STD 66 Unique Resource Identifiers (URIs).[8]
- Identity is a level of indirection used to access a resource *hiding form, location, and function* (emphasis added).
- Relationships between resources shall be expressed using identifiers (IDs) and only through IDs. A change shall be managed whereby a reference to the current version of a Resource is a pointer to an immutable object.

  An immutable object is defined in the JT-NM RA as

> *"a Resource whose state cannot be altered after it is created. If the Resource is changed (i.e., mutated) in any way, it is considered as a new revision of that Resource with an updated revision number component of its unique ID."*

Any change, no matter how slight, in a resources state results in a change in the unique ID for that resource. For example, in **Fig. 1**, if just one audio sample is removed from a piece of content, then the new content must be issued a revised ID.

Names

One could argue that the concepts and best practices around names led to the creation of what may be man's most powerful invention, the internet. The most common form of an internet name is a URL or Uniform Resource Locator.[9] Examples of URLs include www.warnermedia.com and www.bbc.co.uk. URLs are just one form of URIs defined in IETF STD 66.

One could use a URL to retrieve the movie GWTW from an MXF file storage system at the fictitious broadcaster ZBC, as follows.

https://mxfstore.zbc.com/gone-with-the-wind.mxf

This might return a copy of GWTW in an MXF file, but the request is not very specific. We might want to specify a particular rendition or version, using a naming convention established at ZBC.

https://mxfstore.zbc.com/gone-with-the-wind-v2.1.3.mxf

But the power of names could allow us to get specific components of that version. Perhaps, we just want the audio track?

https://mxfstore.zbc.com/gone-with-the-wind-v2.1.3/audio[0].mxf

We could get even more specific, asking for a particular audio sample.

https://mxfstore.zbc.com/gone-with-the-wind-v2.1.3/audio[0]/15182:370000000.mxf

A generic Content Application Programming Interface, or content API, was proposed by Dr. Richard Cartwright in an article titled, "The Agile Media Blueprint."[10] The general form of this content API is:

https://<device>.<domain>/<content>/<stream>/<time>.<format>

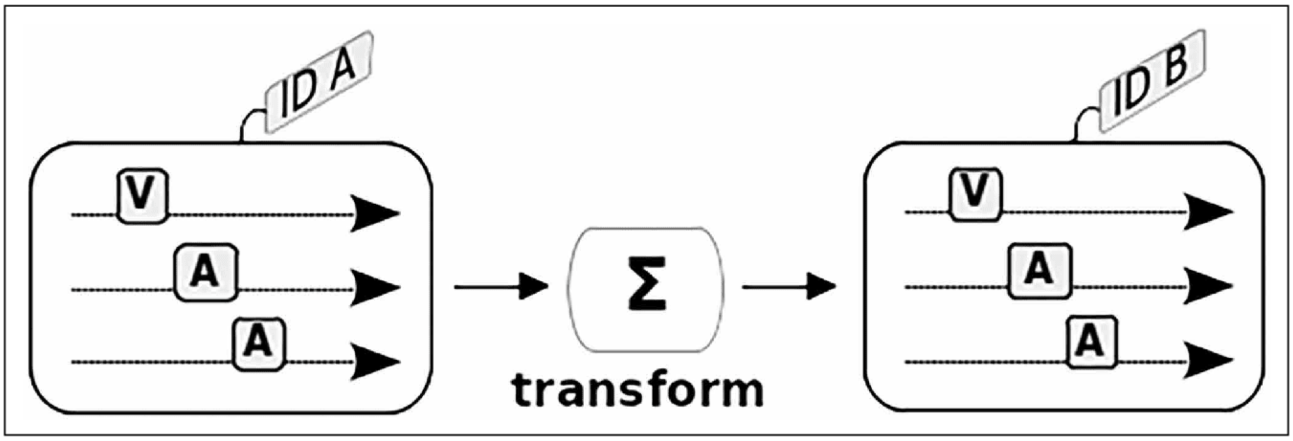

FIGURE 1. Immutable object. An object is immutable when even the slightest transformation, for example, the elimination of a single audio sample, has occurred. This must result in a change in the ID used to identify that resource.

The path parts are:

- <device>—a local name for the device, possibly an alias.
- <domain>—a global or local domain name.
- <content>—subresources that represent complete items of content, with optional extra services that help find the content, such as "search.html" or "content.json."
- <stream>—subresource that represents component parts that can be used to find, synchronize, and process the components such as "cable.json" or "description.html."

This content API could be used as the interface to a media factory that is able to provide content in response to well-formed queries at the interface. As shown in **Fig. 2**, taken from Dr. Cartwright's Agile Media Blueprint paper, the content API serves as the main interface between content creators and a persistent content store in a new kind of media facility. That API may be used with microservices to transform stored content, or even create new content "on-the-fly" in response to queries from social consumers, interacting through the same content API.

If an organization has a common framework for naming resources, if it is possible to unambiguously refer to resources using a unique name as an ID, if the organization enforces the concept of immutable objects, and if the organization prevents end-runs around the system, ensuring that what is returned is what is expected, then that organization may unlock tremendous power in their media systems. Being able to refer to media in an IT-native way using IETF Standards and RFCs is an extremely powerful concept.

A key point about URNs is that they will get you the resource you expect. But there are best practices that should be followed to get the full benefit of these names. You might recall the second tenant of the JT-NM Identity Framework.

Identity is a logical indirection that shall be resolvable to provide access to a representation of a Resource, as per RESTful principles,[11] hiding form, location, and function. Conceptually, physical-resources are not addressed directly.

You use an ID to provide access to a representation or rendition (a specific version) of a resource, but you do not know how that representation is stored, you do not know where it is stored, and you do not know what is required behind the scenes to get you that representation. There is also an admonishment to not make an end-run around the concepts of resources and identity, by, for example, referring directly to "that file on that video server" (addressing a physical resource directly). If you do that, the benefits of this approach unravel quickly.

DNS and DHCP

The domain name service (DNS) and the Dynamic Host Control Protocol (DHCP) are two key technologies that unlock the power of names. We already learned that we can use URNs to refer to resources unambiguously.

What does using URNs get us? If we follow the rules, it gets us a lot. Not only does it allow us to, for example, access content at a granular level, but with DNS, DHCP, and the right infrastructure, it gets us business continuation, the ability to easily take equipment offline for maintenance, a way to share the load across multiple servers with similar capabilities and much more.

A DNS server answers a generic request for a resource with a specific place where a server or other resource can be found that can fulfill that request. Put another way, the DNS server is responsible for telling you where you can find your requested movie. If you have more than one server capable of delivering the movie, then DNS can be configured to distribute requests to the different servers (load balancing) using a number of different algorithms. If one of the content servers should fail, the DNS server can direct all

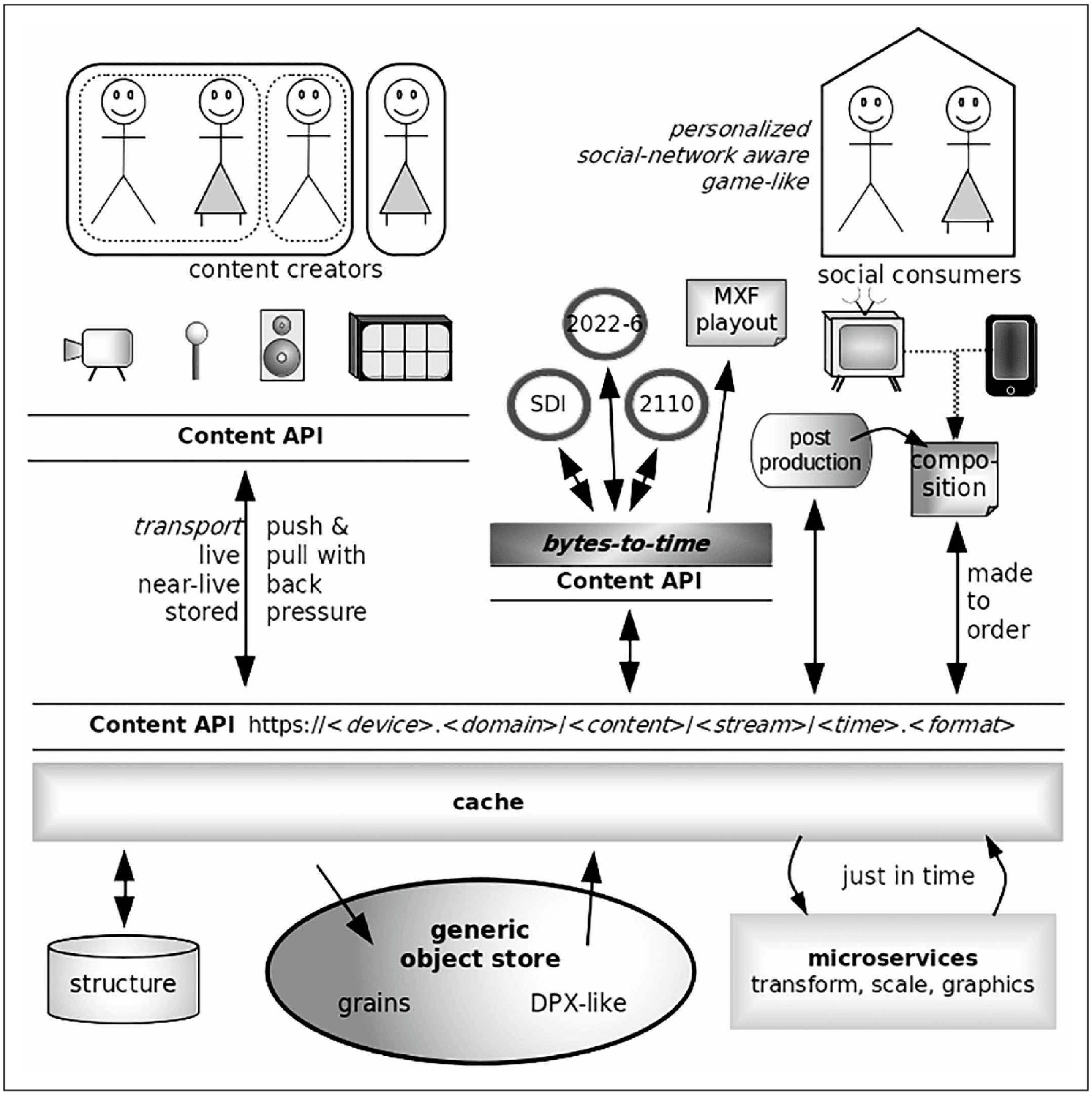

FIGURE 2. In a new kind of facility, a universal content API could be between content creators, social consumers, and a media factory.

requests for the movie to servers that are still working. If you need to perform maintenance on a content server, you can remove its entry in the DNS server temporarily. Any subsequent requests for that movie, or any other movies stored on that server will be resolved to other servers on the network.

This is just one example of the power of names. It is hard to understate the importance of this fundamental concept. But for this to work well, we have to break some very deeply ingrained habits. Let me illustrate with a real-world example.

A national television network had a deal with a major automobile manufacturer. Every week, that manufacturer would announce a sale on a specific model of vehicle. They would send a new commercial to the television network, and on Friday morning, the new commercial would begin to air. On Friday, the Operations Department failed to load the new commercial onto the servers. As you might guess, rather than announcing the new sale item, the old sale continued to be played. Soon the manufacturer called the network reporting that customers were coming in over the weekend demanding that the old sale be honored. Many high-level, very uncomfortable meetings ensued.

How did this happen? The media company failed to ensure that the ID or name of the commercial was unique, and that it should hide its form, location, and function. It turns out that the "name" or ID of the car commercial never changed. And instead of referring to the specific commercial content that should air, the name referred to "that Friday car sale commercial."

As engineers, we care very much about where the content is stored. Because of this, we might want to add an entry in our DNS server so that we can access a specific piece of equipment. For example, https://mxfstore3-1.zbc.com might refer to a specific MXF content server, whereas https://mxfstore.zbc.com/ might refer to a pool of servers. This URL violates the rule that an ID should hide its location. Why is this a problem? If the URL directs a request to a specific server, then DNS cannot reroute the request if "mxfstore3-1" is down. This antipattern destroys the ability to make use of the built-in resilience in DNS. However, a URL that points to a specific server could be very useful for maintenance. This is the beauty of the internet naming system—nothing prevents a resource from having multiple names.

Idempotency

Idempotency is another key concept in IT. An operation is said to be idempotent if you can apply the same operation and get the same result every time. A simple example would be the difference between separate "stop" and "play" buttons, and a single push-button, which toggles between executing "play" and "stop." The first case is idempotent—pressing "play" always sends a "play" command to the device, no matter how many times the button is pressed. The second example is not idempotent, because pressing the toggle button gets you different, and perhaps unpredictable results, depending on the state of the server.

A national television network had a piece of equipment that played commercials. The equipment was programmed to begin a commercial break by playing the first commercial and then playing all subsequent commercials in the break without any operator intervention. Unfortunately, pressing the "play" button performed multiple functions. The first time "play" was pressed, the playback of the commercial sequence was initiated. Hitting the "play" button once the commercial sequence is started causes the equipment to execute a "next" command, dropping the currently playing item and playing the next commercial in the sequence. There was a very small amount of lag between the time the operators hit "play" and when the video first appeared on their monitors—perhaps 125 ms. This was problematic with new operators. If they did not see the commercial begin immediately, they would hit the "play" button again multiple times. This had the effect of initiating playback, immediately sequencing through all the commercials and then causing the machine to pause in black to wait for the initiation of the next commercial break. (More very unpleasant meetings.)

The principle of idempotency greatly simplifies control in IT systems and is one key to enabling the scaling of IT systems.

Stateless Machines

Automation systems and other media control systems are typically designed to know the state of every single controllable device in their domain. If you are a control system, you need to know the status of things you are controlling. But knowing the state of every device comes at a price. You have to store that information. You have to update that information. If you have multiple controllers on your network, there is a burden on controlled devices to respond to multiple requests for status updates, or you have to figure out a way to instantaneously communicate the status of devices to the different control systems and provide ways to realign them if they get out of sync. This problem becomes intractable pretty quickly. If you increase the size of any system, the amount of state information in the system grows—and not linearly.

Many IT systems use a Publish/Subscribe model. In this model, if you want to know the status of a given device, you can subscribe to notifications from that device. The device then publishes its status on a regular basis, or when it changes, and every subscriber is notified of the change. Controllers make no assumptions about the state of the selected device. They track the state by being specifically told of the state of the device through a subscription. They may maintain a state diagram that lists all possible states for a device, and the allowed transitions from one state to another. When the controller receives a notification that a device is in a particular state, it can look at the state diagram to determine what commands are now valid. In a further refinement of this concept, a database may be used to record all notification messages with timestamps. In these systems, it is possible to "playback" the database to go back in time and simulate the states of all subscribed devices, something that can be invaluable in troubleshooting.

Quoting from Fielding's dissertation:[12]

"The client-stateless-server style derives from client-server with the additional constraint that no session state is allowed on the server component. Each request from client to server must contain all of the information necessary to understand the request and cannot take advantage of any stored context on the server. Session state is kept entirely on the client. These constraints improve the properties of visibility, reliability and scalability. Visibility is improved because a monitoring system does not have to look beyond a single request datum in order to determine the full nature of the request. Reliability is improved because it eases the task of recovering from partial failures. Scalability is improved because not having to store state between requests allows the server component to quickly free resources and further simplifies implementation."[13]

(Fielding does point out a disadvantage of this approach in that it may increase the repetitive data sent in a series of requests.)

Eventual Consistency

For many years, computer scientists focused a lot of attention on ensuring that databases were always perfectly in sync. The idea that you could get outdated information from your database or client was an anathema. As systems became larger, and the number of databases and transactions grew to hundreds and then thousands of transactions per second, it became impossible to keep everything in perfect synchronization. Somewhere along the way, someone had an "ah-ha" moment, noting that if some of the other core concepts in this article are in place, then it is better to focus on making sure that databases are consistent over time, rather than blocking transactions or using other techniques to ensure consistency is achieved at all times. This concept is known as eventual consistency.[14] With this new approach, underlying services are responsible for ensuring that the data in these databases is consistent—eventually. However, at any given time, the data held in different clients or databases may be out of synchronization. This may seem like the worst idea ever. But the critical leap in thinking is to look at the user requirements and understand if absolute synchronization at all times is really necessary. It turns out that in many cases, it is not.

Why is eventual consistency important? There are two reasons. The first is that as systems become large (and some media systems are very large indeed) performance can suffer, to the point that the systems stop functioning. The second is that for many applications, eventual consistency allows the applications to scale tremendously with no impact on integrity. If high integrity is required, then strong eventual consistency or even reverting back to the older approach of blocking everything until data is consistent, called *strong consistency* is possible.[15]

Microservices

A microservice is a resource on a network that does one thing well.[16] When you view a typical online shopping application, you are not looking at a monolithic application. Instead, you are looking at the output of anywhere from a few, to hundreds of microservices, integrated into a unified view and presented on your web browser. Microservices may look at the availability of an item, allow you to modify your account, or a microservice may present you with a price for the item. Compare that approach with the typical monolithic application, which may provide hundreds of functions all rolled into one computer program. The difference between these two approaches is shown in **Fig. 3**. On the left side of the figure is a monolithic application with a single database. On the right side of the figure is an application that performs the same function, but it is built around microservices. The difference between the two systems is imperceptible to the end-user. However, the architectures are quite different. In the case of the monolithic application, requests from the user are passed to the single application that prepares responses, which are then presented to the user. In the case of the microservice-based approach, requests are quickly parsed through a pass-through backend, which delivers the requests to the appropriate microservice. Responses are aggregated from the different microservices and then presented as a unified whole to the user by the presentation logic. Each microservice is responsible for its own data. Importantly, each stands on its own, allowing a microservice to be maintained or changed without taking down the entire application. In properly architected systems, capacity may be scaled up by adding more microservice instances in the areas of high demand.

Microservices have become a best practice in the IT industry. The biggest reason is that it provides a way to control complexity. Generally speaking, the complexity of an application grows at the square of the number of functions the application performs. This rapid growth in complexity means that as systems grow, they approach a point where they become very difficult to maintain, hard to modify, and impossible to test. And this can happen very quickly. This leads to instability and unpredictability. Microservices have proved to be a very effective way to address this problem.

There are many other benefits to microservices compared to monolithic applications, including scalability, fault tolerance, ease of upgrades, simplicity in documentation, and more.

Other Concepts

There are other important concepts we should consider but space considerations will not allow us to go into detail. However, we will briefly touch on a few additional points.

Single Source of Truth

Jim Trainor wrote an article dedicated to this topic.[17] It is critically important that we strive to have one authoritative source for every piece of data in our systems. As much as possible, data should not be duplicated. Unfortunately, many media products are built with the view that they own all the data they work within the facility. As an example, a lot of information is duplicated between traffic systems and automation systems. In newer facilities, there is duplication between automation control systems and SDN controllers. These are just two examples. Multiple sources of the truth are the cause of never-ending issues with trust, and in large systems, they can become unresolvable.

Automate Everything!

The IT industry has been making increasing use of automation. These automation tools make testing and

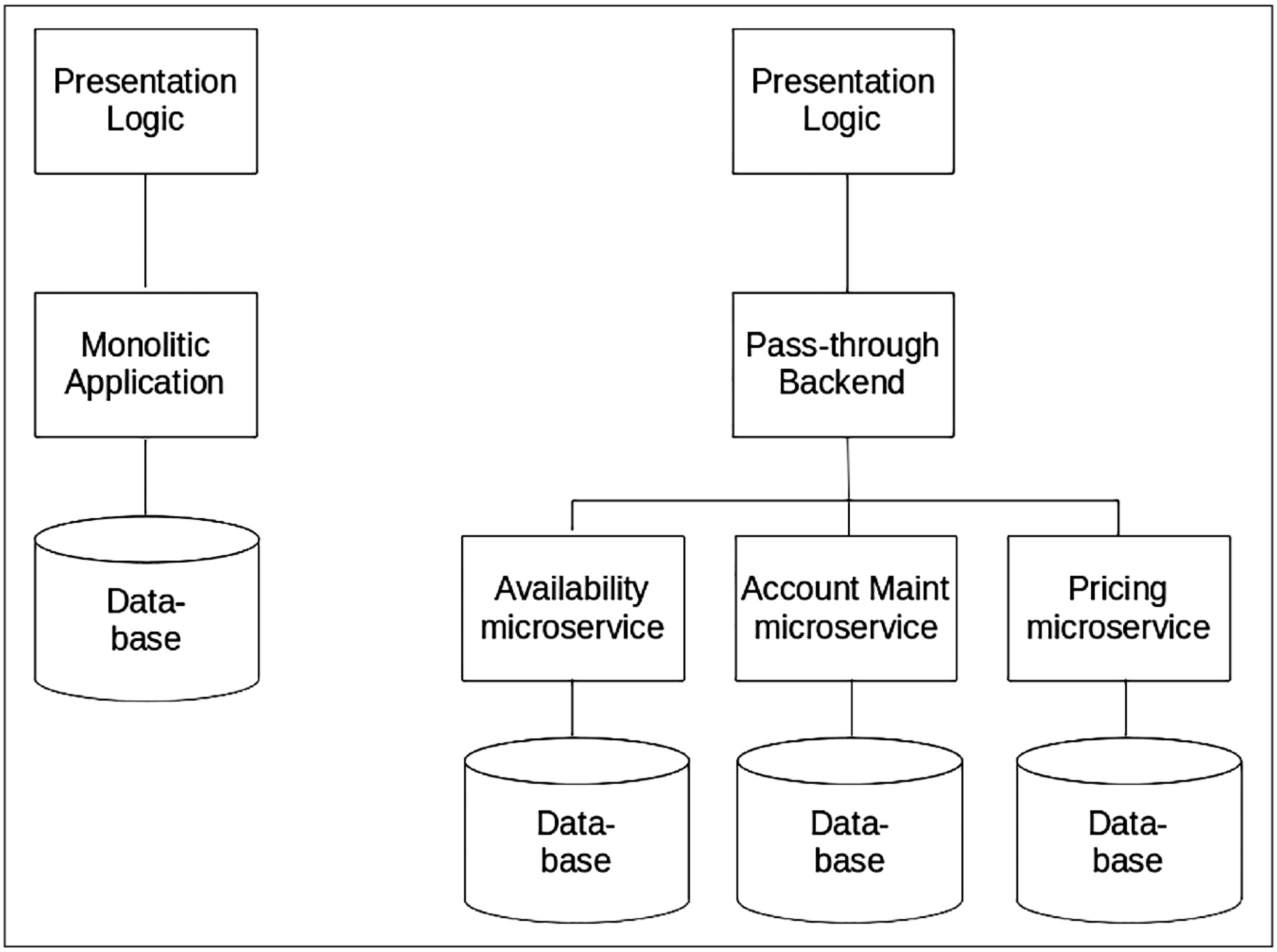

FIGURE 3. Simplified architecture of a monolithic application compared to one designed around microservices.

deployment of code part of the normal software development workflow. One trend that may be of interest to broadcasters who are looking for more flexible facilities is infrastructure as code (IaC). IaC allows facility designers to create, test, and deploy new facility configurations, all under automation and version control. This allows facility engineers to change their facility's "wiring" quickly (think special events such as elections, etc.) or to deploy infrastructure changes during less critical periods such as late at night, with the ability to revert back to the previous configuration at the push of a button. CBC/Radio Canada has been looking at using IT automation tools to configure their new facility in Montreal. They describe their experience in another article in the June SMPTE Journal.[18]

Security

As an industry, we have begun to be much more aware of security as an issue, and the EBU Media CyberSecurity (MCS) Group[19] has done a lot in this area, including testing media applications, providing security requirements recommendations for equipment acceptance,[20] and proposing a security issues notification framework to alert people to vulnerabilities in our industry. But we need to do more to make security one of the first topics we discuss.

Cloud Thinking (for Physical Kit)

There are few if any media companies that can start from a clean slate when moving to IT. Vendors have existing products, many of which may not adhere to the points in this article. What to do? Gordon Castle from Discovery/Eurosport has introduced the concept of employing cloud thinking for a physical kit.[21] Rather than saying, "Transformation is impossible given the physical kit we have to work with," we can think about how this kit might be wrapped to be presented in a more cloud-friendly way, allowing for immutable identity, breaking down access to the physical kit into appropriate microservices, and so on. A physical kit introduces challenges, but if we start thinking about how it might be presented in a more IT-friendly way, we may think of creative ways to incorporate this kit into the software-oriented media facility.

Self-Documenting APIs

For a number of years, there have been efforts in the industry to develop common APIs. (For example, this article highlights the potential of a content API as proposed by Dr. Cartwright.) Some APIs such as the Networked Media Open Specifications (NMOS) APIs for registration, discovery, and connection management have been successful. Other efforts, perhaps less so. And while the idea of a common set of APIs for media seems interesting, a number of factors may keep this out of reach.

It is likely that bespoke APIs will always be a part of media infrastructures. Given this, are there best practices from the IT world we can adopt with regard to media APIs. The definitive answer is "yes." A survey by programmableweb.com[22] found that "developers want clearly documented APIs that are reliably available." In fact, the desire for "Complete and accurate documentation" was rated even higher than service availability or service responsiveness! Well-documented code is clearly a priority. The IT industry has been stressing the benefits of well-written, self-describing APIs for years. And in fact, there has been a robust discussion of what makes a good API versus a bad one for many years as well.[23] As we move to a more IT-centric media world, it would behoove us to learn from the IT world's mistakes with regard to the documentation of APIs.

Well-written self-describing APIs have several characteristics. They do one thing well, they are simple to understand and use, they do not require special knowledge of what is going on behind the interface to implement the API, and they contain information and documentation, and sometimes even tests, that aid developers in creating successful implementations. For a pretty exhaustive discussion of what it takes to make a good API, see "How to Design a Good API and Why it Matters," written by Joshua Bloch, principal software engineer at Google.[24]

Technologies such as Swagger[25] and Integrated Development Environments (IDEs) such as the Eclipse IDE[26] help enforce good behaviors in API development and support "code first" environments where, once the user requirements are understood, developers immediately start writing code to address those requirements. The documentation is essentially developed alongside the code.

Conclusion

This article has introduced a number of key concepts that have helped the IT industry deliver the "ilities" that their users demand (flexibility, scalability, reusability, and so on). We now find ourselves in a position where media consumers are requesting those same "ilities."

A potential way forward would be to carefully consider how we can adopt the bedrock principles in this article that have enabled modern IT and computing. Where possible, we should employ systems that follow these principles, and when we are forced to deploy elements of our systems that exhibit behaviors that are IT antipatterns, we should seek to either encapsulate those behaviors, or accept the behaviors but be keenly aware of the limitations those behaviors may impose on our overall systems, now and in the future.

We may wish to follow best practices from the IT world in terms of automation, including looking at how configuration and deployment of infrastructure might be done as code, providing some of the flexibility and agility our users are requesting.

We may want to consider security at the beginning of our projects.

If we decide to look to IT as a way to help deliver the functionality that is being requested from our users, it is likely that our future will necessarily involve making more use of APIs. We should ask our vendors what they are doing to follow IT best practices in developing reliable, well-written, and well-behaved APIs.

References

1. "How to Thrive in Today's Disrupted Media Markets," Forbes, Sep. 2019. Accessed: Apr. 2022. [Online]. Available: https://www.forbes.com/sites/awsmediaandentertainment/2019/09/12/how-to-thrive-in-todays-disrupted-media-markets
2. Information Technology (Noun), "The Technology Involving the Development, Maintenance, and Use of Computer Systems, Software and Networks for the Processing and Distribution of Data." Accessed: Apr. 2022. [Online]. Available: https://www.merriam-webster.com/dictionary/information%20technology
3. Computer Science, [Online] Available: https://www.merriam-webster.com/dictionary/computer%20science
4. "Ariane 5—Flight 501 Failure," 1996. Accessed: Apr. 2022. [Online]. Available: http://sunnyday.mit.edu/nasa-class/Ariane5-report.html
5. "Joint Task Force on Networked Media (JT-NM) Phase 2 Report—Reference Architecture v1.0," Sep. 2015. Accessed: Apr. 2022. [Online]. Available: https://jt-nm.org/reference-architecture/
6. R. T. Fielding, "Architectural Styles and the Design of Network-Based Software Architectures," 2000. Accessed: Apr. 2022. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
7. "Joint Task Force on Networked Media (JT-NM) Phase 2 Report—Reference Architecture v1.0," "Identity Framework," Sep. 2015. Accessed: Apr. 2022. [Online]. Available: https://jt-nm.org/reference-architecture/
8. T. Berners-Lee, R. Fielding, and L. Masinter, IETF STD 66, "Uniform Resource Identifier (URI): General Syntax," 2005.
9. T. Berners-Lee, L. Masinter, and M. McCahill, IETF RFC 1738, "Uniform Resource Locators (URL)," 1994. [Online]. Available: https://www.ietf.org/rfc/rfc1738.txt.
10. R. Cartwright and B. Gilmer, "Agile Media Blueprint—Creating and Monetizing Content Using the Internet Technology Platform," 2018. Accessed: Apr. 2022. [Online]. Available: https://static.amwa.tv/agile-media-blueprint.pdf
11. Representational State Transfer (REST) Accessed: Apr. 2022. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
12. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures." Accessed: Apr. 2022. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
13. R. T. Fielding, Section 3.4.3, Client-Stateless-Server.
14. "Balancing Strong and Eventual Consistency With Datastore." Accessed: Apr. 2022. [Online]. Available: https://cloud.google.com/datastore/docs/articles/balancing-strong-and-eventual-consistency-with-google-cloud-datastore

15. "Eventual Consistency vs. Strong Eventual Consistency vs. Strong Consistency." Accessed: Apr. 2022. [Online]. Available: https://stackoverflow.com/questions/29381442/eventual-consistency-vs-strong-eventual-consistency-vs-strong-consistency

16. T. LaToza, *Microservices—Design and Implementation of Software for the Web*, George Mason University: Fairfax, VA, 2017. Accessed: Apr. 2022. [Online]. Available: https://cs.gmu.edu/~tlatoza/teaching/swe432f17/Lecture%2011%20-%20Microservices.pdf

17. Jim Trainor, "Single Source of the Truth," *SMPTE. Mot. Imag. J.*, 131(5):23–29, Jun. 2022.

18. Sunday Nyamweno, Patrick Morin, Carl Buchmann, Alexandre Dugas, and Felix Poulin, "Infrastructure as Code at CBC/Radio-Canada's Media-Over-IP Data Center," *SMPTE. Mot. Imag. J.*, 131(5):30–37, Jun. 2022.

19. European Broadcasting Union (EBU) Technology and Innovation, "Media Cybersecurity." Accessed: Apr. 2022. [Online]. Available: https://tech.ebu.ch/groups/mcs

20. European Broadcasting Union (EBU) R 148, "Cybersecurity Recommendation on Minimum Security Tests for Network Media Equipment, Apr. 2018." Accessed: Apr. 2022. [Online]. Available: https://tech.ebu.ch/docs/r/r148.pdf

21. G. Castle, "Eurosport Technology Transformation." Accessed: Apr. 2022. [Online]. Available: https://www.youtube.com/watch?v=QC-649Z6hsI

22. A. DuVlander, "API Consumers Want Reliability, Documentation and Community," Jan. 2013. Accessed: Apr. 2022. [Online]. Available: https://www.programmableweb.com/news/api-consumers-want-reliability-documentation-and-community/2013/01/07

23. "How do you define a good or bad API? Accessed: Apr. 2022. [Online]. Available: https://stackoverflow.com/questions/469161/how-do-you-define-a-good-or-bad-api

24. J. Bloch, "How to Design a Good API and Why it Matters." Accessed: Apr. 2022. [Online]. Available: http://www.cs.bc.edu/~muller/teaching/cs102/s06/lib/pdf/api-design

25. Swagger, "API Development for Everyone." [Online]. Available: https://swagger.io/

26. Eclipse IDE. Accessed: Apr. 2022. [Online]. Available: https://www.eclipse.org/eclipseide/

About the Author

Brad Gilmer is the president of Gilmer & Associates, Inc., and the executive director of the Video Services Forum (VSF) and the Advanced Media Workflow Association (AMWA). Both the VSF and the AMWA received Emmy Awards during Gilmer's tenure. He is a SMPTE Fellow and a first recipient of the SMPTE Workflow Systems Medal. He was previously employed at Turner Broadcasting System, Atlanta, GA, where he and his staff were responsible for engineering and operations for the Entertainment Division Worldwide. He is an author and an editor and has written many articles on computers and networking for media industry publications.

Take the Lead

SMPTE Corporate Memberships lift your organization to the next level

Are you taking full advantage of all SMPTE can offer? We've introduced new Corporate Memberships levels designed for every type of business, whether you're an industry leader, newly minted start-up, small business or consultant. With professional and executive education, conferences and events – plus a full array of opportunities for corporate promotion, standards development and student support – you'll spark success for your whole team.

Review the benefits of corporate membership, and lock in a lower rate by renewing now, before prices increase in July.

SMPTE

Visit smpte.org/corporate-membership

Our SMPTE members are visionaries, entrepreneurs, technologists and engineers, working together to drive the industry forward with technical brilliance and innovation.

SMPTE