Minerva: Applied Software Engineering for XR

Blake A. Schreurs

ABSTRACT

This article describes Minerva (Multiuser Intuitive Exploitation and Visualization), a proof of concept demonstrating a dynamic multiuser alternative to traditional file-oriented intelligence processing and dissemination pipelines. The main goal of the effort was to enable multiple clients to look at the same data in the ways that worked best for them depending on the type of device they were using. Although Minerva did not evolve into a fully developed software product, the team of developers at the Johns Hopkins University Applied Physics Laboratory (APL) created a functional prototype and demonstrated effective use of standards and microservices for hosting and streaming static content. APL teams are applying the lessons learned from this effort to other projects seeking to take advantage of XR to improve traditional data production and representation pipelines.

INTRODUCTION

The Minerva (Multiuser Intuitive Exploitation and Visualization) project, one of the first virtual reality (VR) projects at APL, was an independent research and development effort to explore the possibility and value of a platform-independent, standards-enabled common operating picture that could work in XR on traditional desktop systems, mobile devices, and the web. Much of the development in this project focused on finding effective mechanisms to provide the same data to all devices and users in ways that were best suited to each device in its operational context.

In the operational scenario envisioned for this research, a VR analyst would be supporting a field operative by collating data and disseminating the most important information to the field operative in a live connected system. In this usage scenario, the two users share an objective, but because of their varied roles, they use different devices. The field operative would be more likely to use a smartphone or laptop, both of which require tools to conserve battery and available bandwidth. The VR analyst, removed from the field, would likely have much more computational power and network throughput at their disposal, empowering them to distill available data into actionable information.

This article describes the design of the system, which provided a live, heterogeneous, multimodal, distributed common operating picture. The system was designed to be composed of modular services, allowing the various technological components to be reused when possible. Even though most applications do not share all of Minerva's design demands, engineers can make use of individual components as needed.

SOFTWARE DESIGN

Minerva had an explicit requirement to have various clients that would work concurrently across a broad range of devices. Because of this requirement, the monolithic application approach traditionally used by the gaming industry (which has since given birth to the modern XR industry) would not suffice. Instead, the team took a microservices approach. This design philosophy requires that interfaces between components of the system be well defined, but it allows specific components to be implemented in whatever technology suits the developer (so long as it adheres to the interface).

The VR and the Windows Desktop clients were written in Unity (a game engine commonly used in the XR industry), and the C# scripts were written in Microsoft Visual Studio, the code editor recommended for use by Unity. The web client was written in JavaScript and used the Mapbox library for visualizing geospatial data.

Sever-side microservices were written in C#, in the hopes that there would be potential to share significant amounts of code or libraries between Unity and the microservices. In practice, the disparities in development philosophy between authoring games and web services did not lead to significant sharing between these two codebases.

Open Standards

As stewards of their government sponsors' resources, APL teams have made the use of open standards and formats an integral part of many system designs. As part of the Minerva project, a considerable amount of time was spent analyzing alternatives to determine which portions of the effort could be performed using open-source software tools and technologies. When open-source options were not viable, the team did research to find tools that were generally affordable. The team was able to successfully develop a reference implementation using free/open-source software, with the exceptions of the Unity game engine and Microsoft Visual Studio (two common, and generally affordable, applications for this kind of development).

Types of Data

For this effort, we focused on three classes of data—static data, inconstant data, and ephemeral data—to understand how the data in each class were used, stored, and transferred between systems. Categorizing each piece of data helped shape the technology selection strategy. Table 1 summarizes the three data classes.

Data Discovery and Hosting

Minerva's Content Directory Service provided a centralized HTTP-friendly listing to allow clients to find various kinds of media. This service provided metadata about the content and federated the collections of all known static content hosting services. Clients could use the metadata to look up specific content or to browse content (for example, the service could provide a listing of available car models if a client wanted to locate a car in the world).

Static data were stored on a protected file system, accessed by a web hosting service over HTTP, and delivered as binary data. This portion of code also provided limited transcoding (file conversion) services. There are many equally viable options for this kind of storage, including simple static hosting (as implemented), data warehouse, and data lake. We opted for a web interface because it was well understood by all client software, simplifying integration with other components.

Inconstant data were stored in a MySQL database and accessed via a RESTful web interface serving JSON formatted data. The example problem concerned relationships between entities, so a relational database was appropriate for this effort. Other relational databases would have been more than sufficient, as would have document-oriented databases. Using RESTful

Table 1. Data classes used in the Minerva project

	Definition	Examples	Transmission Considerations
Static data	Data that are expected to remain the same any time they are accessed	3-D models, images, textures, sound recordings	Often large and proper interpretation requires the entire piece of data (no partial streams). Lossy transmission tools are not appropriate. Once the data have been downloaded, they can be cached for future use.
Inconstant data	Data that are prone to change over seconds, minutes, or days	Geospatial data, relational data	Often smaller than static data but require the ability to update or re-query information at run-time and often need more reliable transmission mechanisms than ephemeral data. The need for caching will depend on each application/client.
Ephemeral data	Data that are generated frequently (multiple times a second), used immediately, and then either archived for offline analysis or discarded	Voice-over-IP (VoIP) audio, XR head/controller position and orientation data	Change constantly, and new data coming in do not generally rely on the correct receipt of previous data. Therefore, less reliable transmission tools such as user datagram protocol (UDP) may be appropriate. Caching is not required.

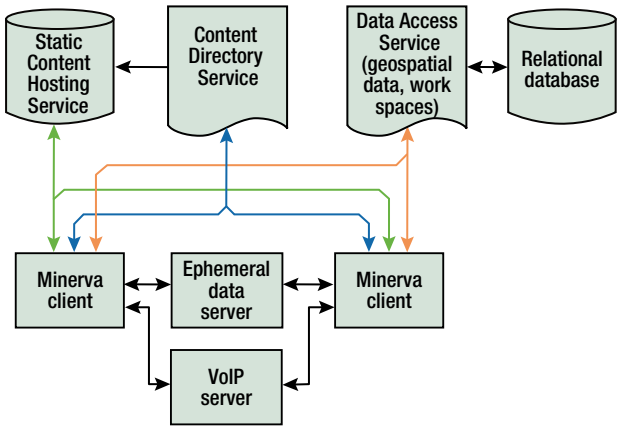

Figure 1. A system diagram showing the major pieces of software used in Minerva. A client requests what entities exist within a work space (orange). For those objects for which the client does not have a representation, the client discovers where it can download an appropriate representation via the Content Directory Service (blue). When processing power and bandwidth permit, the client requests the files from the Static Content Hosting Service (green).

web interfaces enabled both thick clients and thin clients to be able to access the same data by using nearly identical implementations, leading to efficient development workflows.

Ephemeral data were broken into two categories: VoIP data and XR headset/controller data. VoIP data were encoded, transmitted, and decoded using the open-source Mumble library and Murmur server, eliminating the need for developers to define a specification or interface. XR headset/controller data were encoded into JSON and transmitted using SignalR (on projects after Minerva, the development team has been using Socket.IO instead of SignalR because of challenges arising from the release of SignalR 2). See Figure 1 for a system diagram.

File Formats

A quality file-format standard provides a mechanism to interchange data effectively between systems. Much like a web page might use JPEG or PNG files to communicate data about an arrangement of pixels on the screen, various file formats exist to communicate the shape, texture, and appearance of objects in 3-D space. Unfortunately, unlike for the World Wide Web, where there is a large body of well-established and efficient file formats, the formats available for 3-D models are either comparatively inefficient or not universally adopted. Since much of this research was designed to include a broad variety of client systems, the team focused on file formats that had extremely broad adoption, and regarded network efficiency as a secondary concern. The OBJ format uses separate files for object geometry, materials, and (optionally) textures. The geometry and material portions of this format are particularly inefficient, as they store data as uncompressed text. Table 2 describes the file formats supported.

Table 2. File formats used in the Minerva project

Format	Purpose	Notes
OBJ/MAT/JPG	3-D model geometry, materials, and textures	See glTF section
PNG	2-D images/textures	
JSON	Head/hand position data	
WAV	Audio	See WAV section

glTF

glTF (Graphics Library Transmission Format) is a file format specifically designed for graphics data. This format, and the GLB sub-format, is rapidly gaining adoption because it is a royalty-free specification and the defining organization (Khronos Group) has shared much of the source code openly under the MIT License. The format efficiently combines geometry, textures, and animation into a single file, easing hosting of content. These public libraries are still considered prerelease versions (the glTF team has not yet released version 1.0), so they may not be appropriate for many projects. Although this format was not mature enough for use in Minerva, it is maturing rapidly and has many advantages over the OBJ/texture approach.

WAV

The WAV format is uncompressed audio, making it a poor choice for a lot of practical use cases. The popular MP3 codec has licensing issues, leading to challenges on some platforms. Ogg/Theora may provide an efficient and open alternative, but this format has not been broadly adopted. For new development efforts that need to serve static audio files, developers should perform an analysis of alternatives to find a format that meets their specific needs.

Diverse Client Support

One of the key precepts of Minerva was to provide a core set of valuable data and allow each client of the multiplatform system to represent the data in the most appropriate way for their particular device. For example, a web-based client may provide a map, while a VR analysis tool would provide an immersive 3-D landscape. Mobile devices could view data in various formats while also providing their real-world location back to the server. Services were set up to allow more specific representations as network bandwidth allowed. For example, a client might initially represent something with a generic model of a car, but when bandwidth allowed, the client could retrieve more item-specific models or

Table 3. Minerva data representations: bottom-up approach

Representation	Approximate Size	Per Gigabyte of Bandwidth
Text	250 B	4,000,000
Image	1 MB	1,000
3-D model	10 MB+	100

textures to better represent the object. This approach allowed some base functionality when the network was constrained and more enriched experiences in permissive environments.

Bottom-Up Representation

To accommodate the various clients, the system needed to be designed to be conservative with bandwidth, storage, and processing by default and to use media that required more resources only when a client specifically requested it. To meet this design requirement, we opted for a bottom-up representation approach: The core of the system used text-based representation, which is sufficient to draw vital information on a map. If clients wanted media services, they were available to provide images (often around 1 MB) and/or textured 3-D models (frequently 10 MB or larger). With this approach, even modest hardware with a constrained network connection could get vital information, while more robust systems could present higher-fidelity data. Table 3 presents the various data representations, their approximate sizes, and the gigabytes of bandwidth they required.

Standard Entities

To be represented, objects in the environment needed to have a minimum amount of data. This provided a baseline level of functionality for each client, so that over time as heterogeneous data structures were added to the system, an older client not designed for a new piece of data could provide essential functions for that data. For any entity in our system, all objects were required to have a name, location, orientation, globally unique identifier, and authoring information (user/timestamp). With these elements, the team was able to reliably ensure that all tools were able to create, read, edit, delete, and search (by name, by identifier, geospatially, or temporally) as appropriate. Optionally, standard entities could have scale, color, and references to images, sounds, textures, or models stored in a static content hosting service.

LESSONS LEARNED

A number of valuable lessons were learned during this project. First, sensor technology continues to improve at a remarkable pace, while communications technologies have not kept up with that pace. The data throughput of either a 4K camera or a lidar sensor was able to saturate the wireless bandwidth available to our system. As a result, we had to develop a strategy to determine which data were needed at run-time, and which data could be preprocessed and cached on a static content service. For the purposes of this research, the development team had to limit live data to device location, operator pose information, voice data, and images no larger than 1080p.

Lidar data, object geometry, and object texture information were all stored on static content servers. This was deemed an acceptable limitation, as our research primarily focused on interacting with data rather than live processing of sensor output. In the future, having smarter sensors will likely be a huge benefit for these kinds of systems: It would be far more efficient to communicate the location and height of a telephone pole than it would be to transmit a high-resolution lidar point cloud of that telephone pole. This would allow clients to use a static representation of a telephone pole without being concerned about the visual details of each individual pole.

Planning wireless connectivity needs to be performed up front, especially when designing a system for use by a broad variety of client devices (cell phones, tablets, laptops, desktops). The Minerva team used an isolated Wi-Fi network but found that effective ranges of the cell phone's GPS and Wi-Fi signal were nearly mutually exclusive, limiting range and delaying tests. While all devices could have theoretically worked via the open internet, extensive security and authentication features would have been required to safely publish the microservices on the internet, which was well beyond the scope of this effort.

As environments get more detailed and enriched, initial load times for data sets can become untenable for operator use. Systems should be designed in ways that allow for streaming of content over time. This allows operators to interact with vital information while supplemental detail information is loaded.

And, finally, XR control mechanisms continue to evolve. While our research provided cutting-edge interfaces for the time, in the last year the industry has delivered great advancements in hand tracking, gesture tracking, and eye tracking, which would likely inform any new application interface designs.

CONCLUSION

The Minerva project sought to explore the possibility of a standards-enabled common operating picture that could work in XR on various platforms, optimizing the data presentation for each device in its operational context. Although Minerva did not evolve beyond proof

of concept to become a software product, the effort was successful in several ways. The functional prototype was a viable initial foray into using XR in realistic operational scenarios. Even in cases where technology has matured such that code cannot be reused, many of the lessons learned from this effort continue to provide valuable design experience and insight for new tools and technologies. After the completion of this project, various XR and non-XR software development efforts at APL have been able to leverage and reuse many of the standards-based microservices created as part of Minerva (both the ephemeral data server and Content Directory Service have since been reused with minimal modifications), demonstrating how this effort has provided value to the Laboratory well past the project completion date.

Blake A. Schreurs, Information Technology Services Department, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Blake A. Schreurs is a Senior Professional Staff member in APL's XR Collaboration Center. He has a BS in computer science from Davis and Elkins College and an MS in computer science from George Mason University. Blake is a virtual reality (VR) and augmented reality (AR) expert, with a focus on using commodity tools and equipment in novel ways to drive positive impact for sponsors. He has expertise in technology readiness assessment, ideation and design, application development, software architecture, and performance tuning. He has extensive knowledge of interactive and real-time visualization technology, with a focus on improving situational awareness for safety, site protection, and Blue force protection. He is one of the cofounders of APL's Immersion Central, an XR community of practice with over 180 members. His email address is blake.schreurs@jhuapl.edu.