Ensuring Forward Compatibility on Networked Simulations

Chris Kubek
ASTi
Herndon, VA
Chris.Kubek@asti-usa.com

Steve Berglie
ASTi
Herndon, VA
Steve.Berglie@asti-usa.com

Robert Butterfield
ASTi
Herndon, VA
Robert.Butterfield@asti-usa.com

ABSTRACT

Despite years of advances in networking technologies and standards such as Distributed Interactive Simulation (DIS) and High Level Architecture (HLA), the industry still faces interoperability challenges. Over time, legacy simulations can become incompatible with more recently fielded devices. Maintaining forward compatibility between heterogeneous platforms can become problematic, requiring a significant investment of engineering and administrative resources when conducting large distributed exercises between diverse trainers. Maintaining compliance with the latest DIS or HLA standards is one way to plan for change, but these standards have weaknesses that allow disruption of even the best-planned trainer life cycles.

This paper proposes an approach to interoperability and composability centered on an open-source protocol that is, by definition, forward and backward compatible. Because this approach is a cross-platform solution, it can be implemented as a straightforward modification to existing trainer protocols. The proposed approach, initially implemented to allow seamless interoperability for record and playback of communications on distributed exercises, is built from a bottom-up perspective, and directly addresses weaknesses in existing industry standards. By leveraging newer, open-source technologies, this paper presents a solution with an immutable wire encoding, and offers a way to achieve flexibility while maintaining interoperable communication. For a large class of simulations, this approach could dramatically reduce interoperability issues throughout the product life cycle. A specific use-case is demonstrated that illustrates how combining the best elements of DIS and HLA benefits compatibility.

ABOUT THE AUTHORS

Chris Kubek has worked as a Project Engineer for ASTi with a specialization in networked communications and modeling. Experience includes modeling and engineering work on Live-to-Virtual radio bridges (Radio over IP) and a focus on DIS and HLA communications modeling for audio/voice. He has provided support for dozens of simulator programs as well as large-scale distributed exercises.

Steve Berglie is a Software Engineer at ASTi with over 13 years of experience building network and simulation management, testing, and performance analysis tools for both HLA and DIS.

Robert Butterfield is the Chief Technical Officer and a founding member of ASTi. He has over 30 years of experience in Training & Simulation and has been an active member of the IEEE 1287 Standard for Distributed Interactive Simulation Committee since 1993.

Ensuring Forward Compatibility on Networked Simulations

Chris Kubek
ASTi
Herndon, VA
Chris.Kubek@asti-usa.com

Steve Berglie
ASTi
Herndon, VA
Steve.Berglie@asti-usa.com

Robert Butterfield
ASTi
Herndon, VA
Robert.Butterfield@asti-usa.com

INTRODUCTION

A number of communication standards have been accepted within the Modeling and Simulation (M&S) community including Distributed Interactive Simulation (DIS), Testing and Training Enabling Architecture (TENA), Tactical Environment Network (TEN), and High Level Architecture (HLA). This paper presents a solution to reducing the level of effort for software developers and integrators of networked simulations. The solution is not proprietary in nature, and simply an approach for all to consider.

This paper is a story of ASTi's own struggle to create products that easily work with multiple standards but are also maintainable. The paper discusses earlier flawed attempts at mitigating many issues and the lessons learned along the way. The solution—an open-source, wire-level specification—is presented. The challenges and solutions are not limited to radio simulation, but are applicable to distributed systems across Modeling and Simulation.

The paper is written from the perspective of an engineer who has been faced with the real-life issues of integrating various standards and protocols.

THE PROBLEM OF MULTIPLE STANDARDS AND EXISTING SOLUTIONS

As simulations are engineered to support the latest capabilities and standards, they also become part of the distributed training environment. Simulators old and new are trying to train together but may lack a common interface layer. Some trainers are made to use DIS, some TENA, some TEN, others HLA. Even if a standard is agreed upon, there still may be a conflict between different versions of the same standard.

Over time, standards grow and change. To highlight this fluidity, a new DIS revision is currently under ballot, and last year IEEE updated the HLA standard to HLA Evolved. This does not mean that the author is arguing against updating or creating newer standards (in fact, quite the opposite). Rather, it is meant to show that network capabilities are always evolving, and should always be evolving. What may be the latest and greatest today is only current for a limited time, and simulations must have a plan in place to accommodate future technologies.

Several approaches have been adopted to maintain compatibility with evolving standards and simulations including gateways and contractual enforcement.

Gateways

An extensive amount of work has gone into gateways and other forms of middleware. Gateways help translate from one standard to another, allowing simulations to communicate by bridging the common interface layer. Gateways have proven to be flexible and maintainable but also have drawbacks.

Gateways can add latency to the network, and may have their own interpretation of the models being shared. The extra layer of complexity can be difficult to debug and integrate (Dingel, 2002). Distributed events using gateways may also need to cater to the lowest common level of fidelity. Advanced features gained by newer standards are either thrown out or ignored. The result can be a complicated "system of systems" where multiple components are hampered by a disjointed architecture.

Contractual Enforcement

Another solution to maintain compatibility is to impose a contractual requirement that enforces backward compatibility. A contract may stipulate that a new application must support old and new standards, and these requirements are based on need. Older applications must then be updated to support the same technological paradigm. Code changes and technology refreshes are implemented for simulators that were previously DIS-specific to now also use HLA.

HLA combats some of the issues of backward compatibility by a reliance on being "link compatible". This means that changing HLA software should be a relatively painless affair, since the software that uses HLA should remain untouched. This lowers the engineering cost of moving to a new Run Time Infrastructure (RTI), the middleware that runs HLA on a device.

But overall, there is a high cost to the enforcement of backwards compatibility. The technical challenges are often more complex than simply mapping data from one standard to another. Developers must also deal with many protocol-specific features such as evolving timestamps or better location algorithms.

What is the impact of these technical challenges? How do developers best deal with a variety of standards in the M&S community? S*imulations are developed to be agnostic of the communication protocol that they are using*.

For example, those who design the Object Model for a weapon system do not rely on DIS-specific enumerations or formats, but instead design for what is important to that system. Talented engineers are able to abstract the communication interface layer and write software that is Domain Specific. A successful application becomes one that is flexible enough to manage multiple standards while minimizing the impact to their domain and to the distributed environment.

Efforts have already been made to help make more simulations protocol-independent but still focused on being network centric. There are several tool sets that generate software classes suited to the Distributed Environment, while maintaining the accuracy of specific Object Models. The tools help simulation developers create models that do not contain protocol-specific information, allowing easier integration with more complex standards (Metevier, 2010).

While progress has been made in helping developers to create domain-specific models, there is a gap in the ability to connect those models to the interface layer.

An Early Failure: ASTiNet

ASTi's solution to the problem of multiple standards was to create our own in-house protocol for radios. The communication mechanism was designed from the ground up and specially tuned for the ASTi radio environment. The protocol was proprietary, sparsely documented, and artfully named ASTiNet (or ASN for short).

The design goals were sound:

- *Flexibility* – Adding a new field should be easy, and the protocol should be extensible. ASTi is always adding new features and modeling more sophisticated radios. Breaking working software versions with newer concepts or features should be a rare exception, not the norm.
- *Focus on radio simulation, not standards* – This sounds like a simple idea. The company is based upon simulated radios, and that Object Model should be a core value. In other words, ASTi radios should think about radios first, DIS or HLA second. However, prior to ASN, the company's products were already adding DIS-specific notations and paradigms.
- *Support the latest technology* – Important networking concepts like IPv6, peer-to-peer, and Zero-Conf were added. ASN was going to be the future and needed to be in position to take the company there.

ASN was successful in meeting these design goals. The product was built to support multiple standards on the same computer. Small, flexible software components bridged ASN onto a DIS network and a number of different HLA federations. Bridging from ASN to an HLA FOM did not impact the radio software, and newer networking concepts did not disrupt the functional radio simulation.

The software achieved an important concept in design called *encapsulation*. The radio environment was protected from other parts of the code base by being completely isolated. Essentially, ASTi had pushed standards like DIS or HLA out to the very edge of the software. The developers and networking experts who worked on the bridging software could do so without changing the simulated radios.

Unfortunately, ASN had two problems. The first problem was that it was proprietary. Despite our best intentions, a proprietary protocol was frowned upon by much of the customer base. The protocol had the feel of something that was trying to replace existing standards. Explaining ASN to a customer was a delicate exercise, and often led to confusion or rumors. There was a perceived lack of interoperability, simply because ASN was unknown to the rest of the M&S community.

The second problem, in hindsight, was dreadfully predictable: code maintenance. The documentation (that which existed) for an entire in-house communications protocol was weak, especially given that this was intended to lead the company into the future. Though the concepts behind ASN were simple, the implementation was not. A single web page on the development network comprised the entirety of ASN's documentation.

The real impact was that any development on network standards or the radio simulation software required expertise. For a small company, it meant that only one or two engineers could work on those pieces of our product.

These issues became readily apparent when ASTi began to develop a new Record and Replay tool.

ASN: Example of Forward Compatibility Challenges

ASTi was developing Record and Replay (R&R) software for a customer. The intent was to have a relatively simple tool that features ASTi's core product area – radio simulation. The tool would be able to record network traffic, and allow a user to control how the audio is played back.

For example, an instructor would be able to select several radio frequencies from a distributed exercise and play the audio back over a set of loud speakers. Features were to include fast forward, bookmarks, pause and filtering options. Exercises could also be recorded and stored for data analysis. The goal was a simple tool that would be low cost and a 75% solution.

The R&R tool would act like a network logger, capturing packets and messages. The key to having a successful (and profitable) product would be the ability to record across multiple standards. Developing separate tools for DIS or HLA would have been prohibitively expensive. This meant that ASN was an obvious choice to meet the multi-standard requirement, and would play a prominent part in the solution.

The scope of effort quickly became apparent. ASN was too complex to simply capture packets, and the software team creating the R&R tool was unfamiliar with the ASN code base. The inclusion of IPv6 as well as a bid-and-declare process made packet capture difficult without knowing all the specifics of the protocol.

The team was also concerned about the format of the tool's recordings. One requirement was to play back older exercises for data analysis. Changes to the R&R tool or in ASN would be detrimental since, during playback, many of the packets would be misunderstood or at worst never received. Backward compatibility could have easily been lost.

The developers were concerned that not only would the R&R have software bugs, but also that new bugs in ASN could be created. Without a thorough understanding of the protocol, the software team was treading on dangerous ground.

ASTi was not without options, but the scope had jumped from a small effort to a large one. The tool would require careful software design and testing, and the cost was much higher than originally estimated.

The challenge of developing the R&R tool was a symptom of a bigger issue: software maintenance. If creating an application to work with ASN was this difficult, how can the company keep the protocol running for the next 2 years? What happens in 5 years when new networking features need to be added? What happens when there is turnover in the expert programming staff?

It was clear that ASN in its current form would be a problem moving forward. The software team needed a solution that would have more staying power. Simply using the latest and greatest technologies was not enough. ASN was easy for the end user and customer, but not for the software developers who maintained it. A newer communications protocol would be needed.

Plan B: The Addition of a New Design Goal

ASN was going to be redesigned, and maintainability was added to the previous goals. This presented new challenges. How could we reduce complexity without compromising the feature set? How can ASN messages be easily read by software in the present and the future? How can the company use newer technologies without adversely affecting the code base?

(This may sound like a familiar problem to the reader. The growth of distributed computing and networking technology has exploded in the last twenty years and made forward compatibility a bigger issue than ever before. When a software program only needs to worry about itself, code maintenance is smaller problem. When many devices are all trying to communicate, the issue is compounded. Modeling and Simulation, Information Technology, Website Design – all of these technology sectors deal with the problem every day (Henning, 2006; Richbourg, 2008).)

The difficulty was determining a method to provide forward compatibility. How could the company best design a communications protocol that could use the best technology available today, but be maintained to incorporate the best technology of the future? The answer was to have a consistent message format, without restricting how the message was sent.

In developing a new communications protocol, the software team made a distinction between wire format and transportation method. The wire format is the actual encoding of data, the precise way the ones and zeros are set and aligned. The transportation method is how that data is sent.

For example, a classic communication protocol is Morse code. The wire format is the "dots and dashes" used to pass information. The transport method is a

flashlight, telegraph, or radio used to send the code, as well as the speed and tempo of the signal. The definition of the message format is always the same. "Dots and dashes" are always strictly defined for each letter and number; the method of transport can vary.

Relying on a consistent format allowed ASTi to remove complicated concepts like IPv6, TCP/UDP, or peer-to-peer from ASN. Those features could all be constructed with separate pieces of code, placed on top of the protocol rather than embedded into it. ASTi was going to focus on the format of the message itself.

THE NUTS AND BOLTS OF COMMUNICATION PROTOCOLS

The structure and format of data is a key piece of any software development. There is a large amount of information and data within the M&S community, and it follows that there is a range of methods to represent that data. Typically with every data set there is a precise way to store, encode, and share the information.

Because computers rely only on ones and zeros, software uses special methods to represent numbers. How are negative numbers described? How can very large numbers be stored efficiently? How are numbers smaller than one defined? Each method of encoding numbers has benefits and tradeoffs. It becomes the responsibility of the software developer to make the best choice of how to format the data, often working within existing standards and simulations. The format of a database can directly impact the compatibility and usefulness of the data.

The ASTi software group had already experimented with a variety of message formats, but found a preference for Protocol Buffers (Protobufs or PBs for short). Protocol Buffers are an open-source data format created by Google Inc. and used extensively throughout Google's data network.

Protobufs work by first developing a message definition file (the .proto file). This file defines all of the data types and default values to be exchanged. In software, the .proto file is referred to as a schema, and is similar to the Object Model Template (OMT) files used in HLA to define a federation. (The analogy is not perfect; OMT files also include transport method information.) Once a schema is generated, a Google toolkit assists the software developer by creating source code based on the messages and data structures defined.

Protobufs have several advantages[1]. The software development team at ASTi uses multiple languages including C++ and Python. Protocol Buffers are language neutral and work with both of those programming languages, as well as Java. Also, because they are open source, anyone from the software community can generate plugins for other programming languages (Pilgrim, 2008). The result is that the supported language base for PBs is always growing.

In addition to language neutrality, PBs are platform neutral. Protobufs can run on Linux or Windows operating systems. Most data formats are susceptible to problems dealing with 32-bit versus 64-bit hardware. One of the primary problems between the two architectures is that variables with one type of encoding in a 32-bit can change and double in size on a 64-bit system. Protocol Buffers are unaffected by this architecture change.

Protocol Buffers being platform and language neutral is one of the main reasons the company found them so appealing. The software group tried to avoid a steep learning curve, and its simplicity allowed Protobufs to meet that requirement. Because PBs are open source, ample documentation is available online including sample code and tutorials. All of these features fit into the goal of maintainability.

Software engineers have a term for a program that can accept future growth, and for software that can be modified by others without knowledge of the original code. The term is *extensible*. When building a new house it is prudent to include the ducting and vents for air conditioning, even if A/C is not part of the original house. In much the same way, software developers appreciate an architecture that can be extended later with little impact to the existing code.

Protocol Buffers enable messaging that is extensible. Data is tagged and must be decoded by a schema on the opposite end. This allows the messaging to grow over time without the developer worrying about byte placement or properly aligning older messages. Older systems using Protobufs can be updated gradually, and new data fields can be added over time, with little impact to the original software.

Extensible is what the original ASN attempted to be, and the software team had found in Protocol Buffers a message format that would accomplish the goal.

[1] A comparison of other interface description languages such as CORBA, JSON, XML or ASN could easily be the topic of another paper. Such an analysis is beyond the scope this paper, but the reader is encouraged to review some of the documents found in the References section.

THE WAY FORWARD

The software team developed a tool to build .proto files from HLA object model templates. A Protobuf data-messaging scheme was created to work with the radio environment. Eventually, the Record & Replay tool was created to capture and log Protobuf messages as needed.

The newer design goals were achieved, and the code base was much simpler as compared to the previous product which accommodated multiple standards. Multiple software developers could now work with the new source code quickly.

But the biggest result of the decision was the discovery of several new impacts.

While Protocol Buffers play an important role, simply having a proven, consistent wire format has proven a huge value to ASTi. By defining the wire format, software developers could adapt their methods to easily support new features.

Revised Design Goals

Below is a revised set of design goals, incorporating the newer goals along with the original objectives:

- *Flexibility* – Adding a new field should be easy and the protocol should be extensible. Breaking devices with newer concepts and features should be a rare exception, not the norm. Standards will evolve to meet the community of practice, and be prepared to evolve with them.
- *Focus on the Object Model, not standards* –The Object Model is the core value. Standard-specific notations and paradigms should be avoided.
- *Be able to support and encapsulate the latest technology* – Requiring a transportation method, no matter how current or proven, can lead to problems downstream. The data may be sent and received in a multitude of ways, but those should never prevent the message from being understood.
- *Maintainability* – With any set of distributed systems, at least once device is in the interim. Maintenance should be expected to occur incrementally, over time.

These goals are not specific to modeling and simulation. Any system of distributed computing platforms can meet these goals by defining a wire format.

WIRE FORMAT VERSUS API – 5 IMPACTS

The remainder of this paper discusses the impacts of using Protocol Buffers to generate a wire format (this is also referred to as having a *wire specification*). The concepts are contrasted to a system that uses middleware and an API (Application Programming Interface), and the authors have attempted to identify both positive and negative impacts.

A full understanding of what constitutes an API is not necessary for this discussion. For the context of distributed M&S, the important point is that an API gives developers access to a piece of software, and that software gives access to the network. A wire specification simply gives access directly to the network. It is the difference between delivering a letter via the Post Office and hand carrying the envelope to its destination.

While the debate over wire format vs. API is not new to M&S, the use of newer technology for message exchange is. The application of a flexible data message allows for growth without heavy restriction of what can and cannot be sent. The sections that follow feature a comparison of DIS, HLA, ASN, and using PBs as a wire format.

Having a wire format should not be thought of as a magic bullet to distributed simulations or a cure-all to the problems inherent in integrating devices. As with any simulation, the advantages and disadvantages should be weighed appropriately.

Impact 1: Robust Interoperability

There are two important concepts – initial interoperability and interoperability over time. In other words, allowing simulations to easily connect and the ability to keep those simulations compatible can produce a robust interoperability.

Having a wire specification creates an on-the-wire determinism. Participants can join an ongoing exercise with nearly 100% reliability and little ambiguity. Rather than having visibility only through piece of software, a wire spec enables integrators to debug the network itself.

As an analogy, imagine an automobile with a mechanical problem. A skilled mechanic could perhaps determine the problem based on symptoms such as how the car drives, the responsiveness of the engine, or the noises that the vehicle makes. But the best way to determine the problem is to look under the hood and examine the internals of the car itself. This is especially true when the person diagnosing the problem lacks complete expertise of the system. Full visibility

aids in the discovery of problems, and does not require complete knowledge of how the automobile is put together.

Visibility into the network gives opportunity for all players to rapidly diagnose problems, and provides for the creation of a toolset that can help with integration. The effect of a wire specification in DIS led to the production of a wide variety of visibility tools, and in some cases integrators have the option of multiple competing vendors. The end result is more options and more tools to help solve problems quickly.

Once integrated, the evolution of simulations over time starts to affect compatibility. With a defined API, participants can become burdened by changes to middleware versions or implementations. With HLA, the RTI can be purchased from a number of different vendors but those vendors are not able to interoperate. Similarly, different versions from the same vendor may or may not be compatible.

This problem of middleware compatibility is a symptom of any distributed system that uses a defined API versus a wire specification[2]. The multiple vendors adhere to a software interface, but not the ones and zeros that are transmitted onto the network. Any method of data exchange and any wire format can be used by each provider.

The biggest impact that results from middleware compatibility is that all players must agree on both the version and vendor of the software. If a federation wants to gain the advantages of a new middleware version, all federates must be updated. The network must be updated across the board so all players can agree. This can become a difficult problem for devices that have already been approved for training.

The extensible nature of Protocol Buffers allows simulations to be upgraded gradually, and adding data fields to a simulation is simple. Unlike changes to middleware, where most devices on the network must be updated in order to achieve interoperability, simulations can update or add data without impacting the wire format. This is especially useful for the world of simulation where the configuration management of a trainer is tightly controlled.

Another analogy to help understand the differences between an API and a wire specification is that of a number of offices attempting to collaborate on a document. A defined API implies that everyone must be using *the exact same version of Microsoft Word*. If one group needs only to read the document without providing any edits, they still must agree on the version in order to work together. However, a defined wire format would mean that all participants could use any word processing application they chose, on any platform they chose, as long as there is a basic agreement on the ones and zeros of the document itself.

Defining a wire format is not a perfect ending; it will always be possible to break interoperability. Additional capabilities will continue to require software changes. Take a simple example of the timestamp. In order for the timestamp to be useful, all players in an exercise must agree on the meaning and format. Imagine if a federation changes to a new timing mechanism, one with a higher resolution that accommodates for network jitter. In this case, it is not just the meaning of the timestamp that would be changing but also the very format of the timestamp itself.

Without updating, the resulting network would have problems communicating. There are ways to mitigate the impact to older devices, but changing code on those systems would be unavoidable. Altering the schema or .proto file would not be enough. Everything using the older timestamp would be out of date.

Similarly, when using Protocol Buffers, removing a message is not simple. A deprecation mechanism is currently not included. The Protobuf specification allows fields that are 'deprecated' but in the existing implementation, this is ignored. The documentation suggests that future implementations may output a warning if applications attempt to use the data message.

Lastly, since Protocol Buffers are open source they are susceptible to versioning. The implementation may change over time and newer versions of PBs could be created with critical features. While it is unlikely that a new revision will break existing implementations[3], the messages that get sent could look different several years into the future. There are several examples of data formats that at one time were heralded for their simplicity and ease of use. But gradually, more features were added, and the complexity made the software difficult to use, and broke interoperability (Chappell, 1998; Henning, 2006).

[2] TENA provides a rare exception to this by relying on a single distributor for the middleware, rather than having competing organizations each create their own implementation. The result is fewer issues with interoperability, but also no other choices. The merits and tradeoffs of this option tend to be more philosophical in nature, based on the value the reader places on marketplace competition.

[3] The authors' confidence of this unlikelihood comes from a lack of updates to the existing PB spec, the presence of Google as a caretaker and prominent user, and the fact that simplicity was a core design goal from the beginning.

Impact 2: Abstraction of Data Encoding

Each standard within M&S defines its own method for encoding data. In architectures such as HLA, each federation is responsible for how data is encoded. For example, the Navy Aviation Simulation Master Plan (NASMP) spends ten pages in the Federation Agreements Document discussing "data representation." A single attribute, such as the frequency of a radio, can be encoded in a variety of ways.

With Protocol Buffers, common software problems that result from encoding format or byte order are resolved. By using PBs as a core message format, software architecture is simplified since any developer can take the defined schema and easily interface with the simulation.

Before more sophisticated platforms and software languages existed, data encoding was important. Computer platforms relied on data being in a certain byte order. Critical optimizations could be achieved on the then-limited hardware platforms.

The growth of distributed systems and superior hardware, however, has shifted the focus of data encoding. Today's networks derive less benefit from having data arrive with the most significant byte first or last. Instead of small optimizations in processing, simply getting all players to agree on a format is the focus of the problem. Many simulation engineers can tell stories about how a single device with the wrong byte order caused headaches before finally being discovered.

While it is a simple concept to get all players to agree to a data format, in practice each area of flexibility can lead to mistakes. The more settings that are required, the easier it is for the talented developer to make an error. Data encoding in a distributed environment is no different than the formatting of this paper. Font size, margin and column length – these decisions do not determine the impact of this individual paper. However, the importance of having a consistent format that matches other technical publications cannot be overstated.

Impact 3: Network and CPU Optimizations

The third positive impact that was discovered was a reduction in network bandwidth. Perhaps it is not surprising that a Google product has been optimized to minimize network traffic. Protocol Buffers focus on sending as few bits as necessary over the wire. The result is a streamlined message, without bloat, that is ideal for over-the-network transmission.

Additionally, Protobufs are constructed for easy deserialization. This means that when a PB message is received, the processing it takes to decode the message and turn it into useful data is minimized. Protobuf messages, especially in C++, are optimized to have low overhead especially when compared to other message formats.

Impact 4: Loss of Service-Based Infrastructure

A negative impact to using Protocol Buffers and, conversely, the greatest benefit of a defined API is the addition of network and simulation services. By using a specialized piece of software, federates are able to take advantage of a sophisticated feature set. Essentially, the defined API allows middleware to do the 'heavy lifting' on the network. These services can provide optimizations to reduce network bandwidth or provide enhanced event ordering.

HLA-based services such as Declaration Management, Time Management, or Data Distribution Management could be lost as a result of a new wire specification. Implementing these services over the network, but with a constrained messaging format is a major undertaking. Likely, a new RTI would need to be created using Protocol Buffers as the message format of choice.

These services are not prohibited by a wire-level specification, but the level of effort required to implement them on top of an open wire-level spec is non-trivial. Vendors would still have flexibility in their implementation, since using Protocol Buffers truly is only a data encoding and the transport method is not defined. One criticism of a defined wire format is that it will restrict a developer's ability to implement the network services (Granowetter, 2003; Woodyard, 2004). Data encoding alone, however, does little to restrict the algorithms needed for network services. The larger problem is that a completely new implementation would need to be created.

Additionally, these services could reduce or eliminate the benefits discussed earlier. Network bandwidth and processing costs could increase to higher levels when a service overhead is added. Sending a simple message from one computer to another has been optimized for network bandwidth when using Protocol Buffers. But when messages and functions increase in complexity, overhead is created, and more network bandwidth must be used.

Impact 5: Lower Barrier for Entry to the Network

Twenty years ago using software to create network traffic was not simple. It required many levels of expertise, and Operating Systems were still in the stages of relative infancy. Creating a UDP packet and

transmitting data on the network was hard work. Common software models and routines were not available to the everyday developer, and networking was a sophisticated concept (Henning, 2006).

At ASTi, the products sold in the 1990s had an Ethernet driver that was developed in-house. Running DIS meant needing absolute control over the software that placed packets on the network and any compromises to that control were unacceptable. In the present day, developing network applications can be trivial. Novice software developers can generate reliable code that transmits data to and from the network with ease.

Superior hardware and sophisticated operating systems have lowered the cost of entering the network. Conducting traffic for distributed applications is now easier than ever.

The final impact of a wire-level format is the advantages that a simpler solution provides. Large programs and simulations will continue to have the resources to work across multiple standards and become inter-connected. Especially since the resources and engineering effort available to those simulations can accommodate a certain level of complexity. But the simplification of a wire-level format allows developers on other programs or simulations to enter the field of play.

As the M&S community continues to grow, the community is getting smaller. Think rapid development, rapid distribution, rapid training. If end users are training on mobile devices like tablets and smart phones with applications developed by another solider, the community cannot afford too high an entry cost.

Serious games, developed by software teams with roots outside of M&S are increasingly popular. Gaming has been established as another piece of the training portfolio, especially suited to exercises where integration time and budget must be kept to a minimum (Atherton, 2009). If serious games are being developed to run on computers at home or abroad, the expertise required for entry should be kept to a minimum.

To date, game developers have been slow to adopt the networking standards within M&S (Thorpe, 2010). While a variety of reasons can be attributed, one of the main issues is the level of effort for integration into a distributed network. Many games already deal with massive multiplayer environments, but adding in defense-specific requirements begins to build a higher and higher barrier.

The level of effort associated with joining a distributed federation is seen as a barrier that is too high to climb, especially for gaming companies that already have a proven revenue stream from the commercial world. By having a wire-level specification, it allows software developers to create their own applications and place them on the network.

Rather than creating barriers to entry, having a wire-level specification creates accessibility and long-term usability to new or smaller simulations. Any platform that supports the exchange of messages can be accessible. This is critical to success of distributed training.

A Comparison Of Network Architectures

The table here depicts many of the tradeoffs between the different architectures that this paper has discussed. Overall, the use of Protocol Buffers attempts to strike a balance between DIS and HLA by being easy to use but still allowing for flexibility in the data transmitted over the network. (The PB column of the table references ASTi's internal implementation of Protocol Buffers rather than an actual standard or architecture. Though the comparison is apt, any use within the community is likely to look quite different.)

Table 1. Network Architecture Comparison

	DIS	HLA	ASN	PB
Simplicity	Very Good	Weak	Good	Good
CPU/Memory Efficiency	Good	Average (Varies)	Avg.	Good
Extensibility / Flexibility	Weak	Very Good	Avg.	Very Good
Compatibility / Visibility	Good	Weak	Weak	Good
Network Services	Weak	Very Good	Avg.	N/A

ASN, also referenced in the table, is included as an example of how a well-planned architecture can become problematic, even for the organization that created it. The original goals of flexibility and compatibility were eventually hampered by both technology and a lack of documentation. Because the architecture was not open, the weak interoperability of ASN had become a major detriment.

DIS achieves many of the goals depicted, but also lacks a critical need – the ability to extend and grow the data on the network. Requiring a new version of DIS every time a new field needs to be added is a burden that many federations cannot cope with. Some simulations have even taken to using a generic

Comment PDU to send massive amounts of data since no other option exists within the standard.

By design HLA is able to accommodate newer data fields with ease, but does not allow for visibility of the network. The architecture allows fully-extensible networks to grow by adding items to the Federation Object Model yet can suffer from its own complexity. The following image encapsulates the core differences between HLA and the use of Protocol Buffers as a defined wire format.

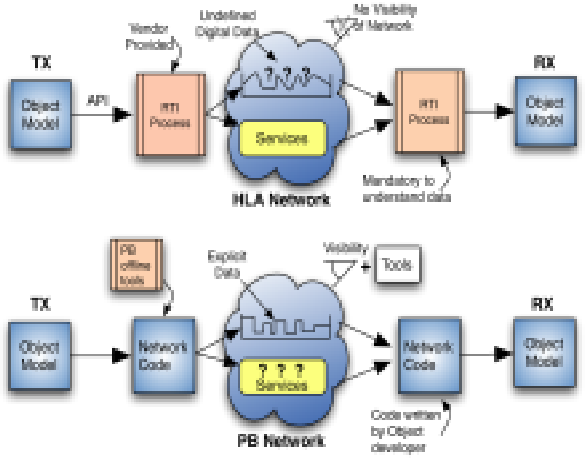

Figure 1. API Defined Network vs. PB Wire Format

In the case of HLA, developers use the API calls provided by the RTI, and must link and compile code against it. In the case of Protocol Buffers, an open-source toolkit assists the software developer with building code.

Additionally, one of the advantages of DIS and its own published wire specification was that engineers could create tools to help with debugging or managing the network. As discussed earlier, simulation developers could either build their own toolset or purchase third-party tools to help with integration. Having a troubleshooting tool is not expressly prohibited by other standards or architectures that use middleware, but the connection to the network is obscured. In most cases, the same vendor that provides the middleware must also supply the toolkit.

CONCLUSION

Simply put, engineering distributed systems is difficult. Simulation developers face a variety of challenges on every federation ranging from semantic disagreements to differences in the databases themselves. Multiple networking standards within the M&S community add to the complication. This paper has attempted to show one solution that can reduce the complexity of interoperability. By simplifying a portion of the networking problem, integrators can spend more time focusing on the other issues.

ASTi's own journey with the issues of multiple standards led to creating an in-house protocol that was burdened by the very complexity the company was trying to reduce. Code maintenance was hampered by a robust set of networking features that ended up being a distraction rather than a benefit.

The solution—to use Protocol Buffers as a defined wire format—was an enormous benefit to the company. Defining an extensible, open wire format achieves balance between current standards by allowing flexibility within the messages that are sent over the network. Similar to how HLA federations define an Object Model for the entire network, the solution presented allows developers to generate the messaging for each and every federation and not require an update to the standard in order to add a new data field.

There were other benefits, including the abstraction of data encoding and a reduction of networking bandwidth. The solution allows developers to spend less time worrying about byte order, variable types or the alignment of bits. Simply use the data structures provided by the Protocol Buffers and go on to address other networking problems.

Rather than focusing on network services (e.g. Data Distribution Management or Time Management), a wire format favors simplicity in implementation over sophisticated features provided by services. Services are not prohibited, but the effort to place them on top of the network wire format is significant.

Finally, systems using Protocol Buffers as a wire format can maintain forward and backward compatibility much more easily, hopefully in a way that allows various simulations to connect and start training quicker. By lowering the barrier to entry on the network, training can be conducted with less integration effort, less engineering, and less cost.

It is not intended to be a panacea to all network woes, but rather to serve as another option. The reader is encouraged to evaluate the solution as an alternative to directly embracing current standards and against his or her own requirements.

Though standardizing a wire format is not a new topic within the M&S community, this paper has attempted to show that by using newer technology, flexibility can still be achieved. Distributed simulations can evolve according to their own requirements, and networks can move forward by being accessible, composed, and dynamic.

REFERENCES

Atherton, E., & Baxter, H. (2009). "Positively Gaming the System: A VBS2™ Training Case Study." *Proceedings Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, November 2009

Chappell, D. (1998). "The Trouble with CORBA." *Object News*, *May 1998*, Retrieved 15 June 2011.

Dingle, J., & Garland, D., & Damon, C., (2002). "Bridging the HLA: Problems and Solutions," *Sixth IEEE International Worskshop on Distributed Simulation and Real Time Applications (DS-RT '02),* Forth Worth, Texas, 11-13 October 2002.

Granowetter, L. (2003), "RTI Interoperability Issues – API Standards, Wire Standards, and RTI Bridges," *Proceeding of the European Simulation Interoperability Workshop,* 03E-SIW-077, 2003.

Henning, M. (2006). "The Rise and Fall of CORBA," *ACM Queue (Association for Computing Machinery),* pg. 4 (5), 30 June 2006

Metevier, C., & Gaughan, C., & Gallant, S., & Truong, K., & Smith, G., "A Path forward to Protocol Independent Distributed M&S," *Proceedings Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, November 2010

Pilgrim, M. (2008), "Protocol Buffers: The Early Reviews Are In," *Dive Into Mark*, 12 July 2008, Retrieved 10 June 2010, Archived by WebCite http://www.webcitation.org/5zLCBq4Ub

Richbourg, R., & Ceranowicz, A., & Lutz, R (2008), "My Simulation is from Mars; Yours is from Venus," *Proceedings Interservice/Indusry Training, Simulation, and Education Conference (I/ITSEC)*, November 2008

Thorpe, J. (2010), "Trends in Modeling, Simulation, & Gaming: Personal Observations About The Past Thirty Years and Speculation About the Next Ten," *Proceedings Interservice/Industry Training Simulation, and Education Conference (I/ITSEC)*, November 2010

Woodyard, J., & Mullally, K. (2004), "Open Run-Time Infrastructure Protocol Study Group Final Report," *Proceedings Simulation Interoperability Workshop Fall 2004*, 04F-SIW-018, Fall 2004