Interoperability and the OpenDOF Project

Bryant Eastham
President and Technical Committee Chair, OpenDOF Project
bryant.eastham@opendof.org

Introduction

In the late 1990's a start-up called emWare formed the Embed The Internet (ETI) Alliance. This alliance focused on bringing the benefits of the Internet to small 8- and 16-bit microprocessors, what would now be termed the Internet of Things. One of the guiding principles for the architecture proposed by emWare was interoperability. The technology produced by emWare was acquired by Panasonic in 2005, extended and augmented internally for a decade, and finally released as an open-source project in 2015 called the OpenDOF Project. The project manages the results of hundreds of man-years of effort consisting of specifications and implementations in a variety of languages.

Interoperability remains one of the five guiding principles of the OpenDOF Project as outlined at https://opendof.org. With 15 years of experience it is fair to ask what we have learned regarding interoperability. This paper briefly summarizes our lessons learned, and more importantly how the principle of interoperability has affected the project.

Interoperability is, unfortunately, a vague term. For the purposes of this paper a set of systems that are "interoperable" satisfies the requirement that any action initiated by a requestor in the system will be understood equally by all potential providers of the functionality. This principle will be probed as it affects syntax, semantics, object model, and security. Finally the paper will present implementation aspects of interoperability. We will constrain the discussion to systems focusing on network communication.

Syntactic Interoperability

The topic of syntactic interoperability is as old as computer networking. As soon as more than a single bit of information needs to be communicated the issues of ordering and field boundaries must be overcome. It is obvious to us today that interoperability at this basic level requires specification and adherence to those defined standards by all parties involved.

It is interesting that we require this level of specification at lower levels, but are often willing to soften our expectations the higher up the stack we move. We see this softening in the following areas:

- Acceptance of proprietary standards.
- Open, but weak standards.
- Standardization through implementation.

Each of these is an attack in one way or another on syntactic interoperability that should be resisted. Syntactic interoperability is the bedrock upon which all other interoperability is built. However, while it is necessary it is not sufficient. If it were sufficient then the bulk of all systems in the world would be interoperable simply because they agreed on how to pass different sizes of integers or floating-point numbers. Clearly there are additional requirements.

One of the most important released artifacts of the OpenDOF Project is a complete and authoritative set of protocol specifications. Engineers use these to determine implementation correctness, and any ambiguity that is discovered is considered a defect in the specifications. Over our 15-year implementation history many different teams have independently implemented the specifications and then verified that their different implementations interoperate correctly. There are currently three separate development lines (two in C, one in Java) and a C# implementation that was originally converted from Java. The benefit of having authoritative and open specifications cannot be overstated, and we would propose such specifications be an industry requirement.

Semantic Interoperability

Once the syntax of the information is agreed upon we must move on to the meaning of that information. Most people intuitively grasp the concept of semantics based on their own spoken communication. Humans excel at communicating in an often weakly defined context. Computers, on the other hand, are totally dependent on strongly defined context.

When this context is not adequately defined, today's systems fail - often catastrophically. Even relatively small mistakes, like a misunderstanding in the unit of a value, can have major consequences. It is critical then that any systems that hope to interoperate must be able to map the meaning of the information they exchange.

The early history of the OpenDOF Project architecture saw the creation of many different bridges to other ecosystems. This experience resulted in the following:

- The use of strongly defined types, along with the ability to extend the type system at the same level as the primitive types.
- Public, well-defined definitions for capabilities and supported interactions, defined with a strict schema.
- Avoidance of "vendor extension" that is not subject to the same level of requirements as the rest of the definitions.

The OpenDOF Project has created an extensible repository project that can store definitions in a variety of different schemas. The hope is that the industry will learn to expect this level of open definition and that, short of standardization, at least we can focus on the ability to convert between ecosystem-specific definitions as an aid to interoperability.

Object Model Interoperability

The term "object model" as used here defines the techniques used to identify or name objects and capabilities. While this may seem a trivial detail, it can and has become a roadblock to interoperability.

For example, consider a system that uses transport information, like IP addresses, as an integral part of object naming. What happens when two disparate systems that use the same private IP ranges must interoperate? Even systems that use extremely flexible naming solutions can also run into problems when they must interoperate on limited transports with small packet sizes.

The OpenDOF Project has dealt with these concerns by adopting the following principles and capabilities:

- Globally unique naming that can be based on any form of unique information.
- Removing all transport information from the core APIs and protocols.
- Providing for registration of capabilities in order to provide open access.
- Discovery of objects and their capabilities.
- Allowing for aliasing of naming information on the wire to minimize packet sizes.

Each of these affects interoperability at some level and should be considered for any system that focuses on interoperability.

Security Interoperability

The preceding sections are often discussed in the context of interoperability. The remaining sections are less often discussed, yet equally important.

The definition of interoperability that we presented in the Introduction referred to a common understanding of an action between a requestor and potential provider. In secure systems this common understanding must extend to the security configuration of the system(s) involved.

As an example of this issue, assume that Alice and Carol are using different systems that wish to interoperate through Bob. Alice and Bob will communicate, and Bob and Carol will communicate. The complexity is that the security configuration for Alice and Carol may not be shared, and without some level of sharing there can be no truly secure communication between the two. Bob can certainly broker the communication assuming that his configuration exists in both systems, but he becomes the weak link in the

trust chain. It may even be impossible for Bob to adequately express the context of a request because the concept of 'Alice' may have no meaning to Carol.

The concept of "end-to-end security" also places important restrictions on interoperability, to the extent that it may require an external security broker separate from both interoperating systems.

The OpenDOF project is designed with security as a core principle, and facilitates security interoperability in the following ways:

- Cleanly separating security in the protocol stack, isolating the concepts of identity, encryption, and access control.
- Focusing on security between peers rather than end-to-end. This has practical implications for complex topologies, and also provides interoperability benefits as the security implementation can change at gateways if required.
- Providing security through a security broker, or Authentication Server, rather that strictly between peers. This also has benefits in general, and allows for any set of endpoints to use the broker - even if they are part of different systems - to accomplish end-to-end security.

There are no easy answers to security interoperability. We believe that the principles and architecture provided by OpenDOF strike a reasonable balance between many competing requirements, while simultaneously providing many unique benefits in terms of flexibility, implementation size, and capability.

Implementation Interoperability

Implementation choices can have an impact on the ease of achieving interoperability between systems. While many of the previous sections can be understood theoretically, implementation issues are often only learned through experience.

There is not enough space to illustrate all of the implementation issues that we have identified, so we will focus on two.

The first has to do with callbacks, a very common programming technique for creating APIs. Consider the following two pseudo-code examples of a callback to determine the value of a particular object that may result from some request elsewhere in the network.

```
Value get( Object O ) {
        return 10;
}
```

```
Void get( Request R, Object O ) {
        R.respond( 10 );
}
```

The example on the right is better for interoperability. Why? Imagine that the particular callback is, in fact, going to make a call to another API to actually determine the value. In the example on the left the callback cannot easily delegate the request without linking the callers thread to the other API. In the example on the right it is easy to pass the processing to another API simply by passing the 'Request' along.

Another example of API support for interoperability relates to security. OpenDOF APIs allow for hooks around the data value buffer passed at the provider and the requestor on both ends of the communication. This allows for the endpoints to independently determine additional security protections (hashes, or full encryption) that can easily be applied to provide for complete end-to-end security. This kind of processing is often buried in other systems making it difficult or impossible to provide such a capability.

Conclusion

Interoperability is the end result of a set of decisions affecting everything from system architecture to implementation. The statement "if it were easy, everyone would do it" applies particularly to interoperability. Unfortunately, many ecosystems do not have interoperability as a goal, and industry has been slow to recognize its importance. The benefits of interoperability are worth the effort, and ecosystems that focus on achieving it deserve recognition.

The OpenDOF Project considers interoperability a core principle. It has had a huge impact on its design and implementation. We provide an open repository where semantic definitions of all kinds can be shared and provide the system for others to do the same, all to increase interoperability. We look forward to cooperating with other ecosystems to achieve this important goal.