PNNL-SA-27035

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

MS-SPEAK Phase 1 Technical Report

GMLC 00068

October 2017

WJ Hutton

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract **DE-AC05-76RL01830**

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <http://www.ntis.gov/about/form.aspx>
Online ordering: http://www.ntis.gov

This document was printed on recycled paper.
(8/2010)

PNNL-SA-27035

MS-SPEAK Phase 1 Technical Report

WJ Hutton

October 2017

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

Abstract

This final report includes information about the technical requirements, technical implementation, field test results, and future work for the MultiSpeak® - Secure Protocol Enterprise Access Kit (MS-SPEAK), which was funded as Grid Modernization Laboratory Consortium as project GMLC00068.

Summary

This summary assumes you are already familiar with GMLC, and the MultiSpeak® standard. The MultiSpeak – Secure Protocol Enterprise Access Kit (MS-SPEAK) was proposed on 9/14/2015. MS-SPEAK was funded in the spring of 2016. Phase 1 of the MS-SPEAK project was kicked off in April of 2016 and was expected to run until 3/31/2017 followed by a "go/no-go" decision for Phase 2 funding and work. PNNL's industry partner, NRTC was to contribute a $50,000/year cost share, which may be the only GMLC project with industry cost share. Unfortunately, it took approximately 11 months for PNNL to contract with NRTC in order to execute the MS-SPEAK project. This severely limited both companies' ability to collaborate on the scope of work for MS-SPEAK.

NRTC was able to participate in the technical requirements phase of the project (Milestones 1.0 and 1.1). PNNL evaluated multiple COTS XML solutions before choosing to use Stylus Studio, primarily because it is not proprietary and is based on Java, which prevents "vendor lock-in". PNNL focused on repeatable work that could add value regardless of the direction NRTC would ultimately guide is in terms of what MultiSpeak servers, end points, and methods we should focus on that would have the widest applicability across the electric industry. After converting MultiSpeak v3 message headers to v5.04, we next focused on completed an "end-to-end" workflow that included message validation, transformation, and logging. In parallel, our team also worked on handling encryption and decryption to support v4.x of the MultiSpeak standard. We exhausted our primary funding while waiting for a contract to be put in place with NRTC. We did reserve a very small amount of funds for three important tasks; field testing, writing this final report, and tech transfer with NRTC.

Our technical approach only relies on Stylus Studio to create XSLT documents, which are an open computer science standard. XSLT documents define how XML documents can be transformed or changed based on syntax. The rest of our implementation is Java-based. PNNL was dependent on NRTC for guidance on which MultiSpeak messages would have the most industry impact if translated, as well as how best to translate the individual fields in those messages, because v5 of MultiSpeak is not backwards compatible with previous versions of the standard.

Field testing consisted of evaluating MS-SPEAK with synthetically generated MultiSpeak traffic based on completed XSLT files, as well as production MultiSpeak messages from one or more utilities in North Carolina. Field-testing revealed several integration issues, which were likely caused by the sequential nature of how the work was performed based on contracting difficulties.

The end result is the evaluation of multiple XML technologies, as well as an implemented proof of concept that demonstrates how to validate and transform MultiSpeak messages between two different standards. Next steps would be to study which MultiSpeak "servers" and methods should be targeted for the creation of additional XSLT files that we hope would provide an 80% solution for any utility to use in their own efforts to integrate multiple MultiSpeak standards, with the utility or vendor providing the remaining 20% effort, which is likely unique to their own situation and not reusable by other utilities.

Acronyms and Abbreviations

ESB+	Enterprise Service Bus
HTTP	HypterText Transport Protocol
MS-SPEAK	MultiSpeak® - Secure Protocol Enterprise Access Kit
NRECA	National Rural Electric Cooperative Association
NRTC	National Rural Telecommunications Association
OMS	Outage Management System
SME	Subject Matter Expert
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
WSDL	Web Service Discovery Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

Contents

Figures

1.0 Introduction

MultiSpeak v3 has been widely deployed throughout the electric industry since 2005. Many software developers have misapplied the MultiSpeak standard in an effort to implement it for various utilities by selecting the wrong use cases and end points, or overloading the standard. A gross example would be to transmit a value for voltage as comments while MultiSpeak makes many different methods for communicating voltage available. The challenge is selecting the correct method and implementing it in a way that adheres to the standard

MultiSpeak v5 did not have backwards compatibility as a design goal. Therefor it is not interoperable with v3 or v4.x—stranding users at their current version. Furthermore, this greatly increases the upgrade cost, requiring that all components that use MultiSpeak be upgraded to v5 at the same time.

PNNL along with NRTC as their industry partner proposed to create an enterprise service bus to validate, translate, and forward MultiSpeak messages to enable interoperability between MultiSpeak standards. Section 2.0 of this document describes the technical requirements we developed for the MultiSpeak ESB+, known as MS-SPEAK. Our technical approach to implementing MS-SPEAK is outlined in Section 3.0, Future Work can be found in Section 4.0. Section 5.0 documents the results of our field testing of MS-SPEAK with NRTC in North Carolina on October 18, 2017.

2.0 Technical Requirements

Perhaps the easiest way to think about the MS-SPEAK ESB+ (Enterprise Service Bus+) is as a "benevolent man-in-the-middle". The ESB+ will leverage the existing NRTC WSR (Web Service Request) prototype to provide two key functions:

1. MultiSpeak specification transformation
   (e.g., v3 -> v4.1, v3 -> v5, v5 -> v3, etc.)
2. MultiSpeak specification validation

MultiSpeak transformation will improve interoperability, extend the life of previous utility investments and deployments, and improve the adoption of the MultiSpeak v5 specification. MultiSpeak validation will be useful to both utilities and vendors.

Adoption of MS-SPEAK is important to us, so we will be using the tools we develop to facilitate the implementation of MS-SPEAK. A community of interest is planned to reduce the redundant work associated with implementing any particular transformation between specifications for various platforms (i.e. vendors) and "end points".

The remainder of this document explores the technical requirements necessary to implement MS-SPEAK, initially at a very high level, then with increasing detail.

2.1 High-Level System Design

Figure 1. Existing MultiSpeak Functionality

End Point$_1$

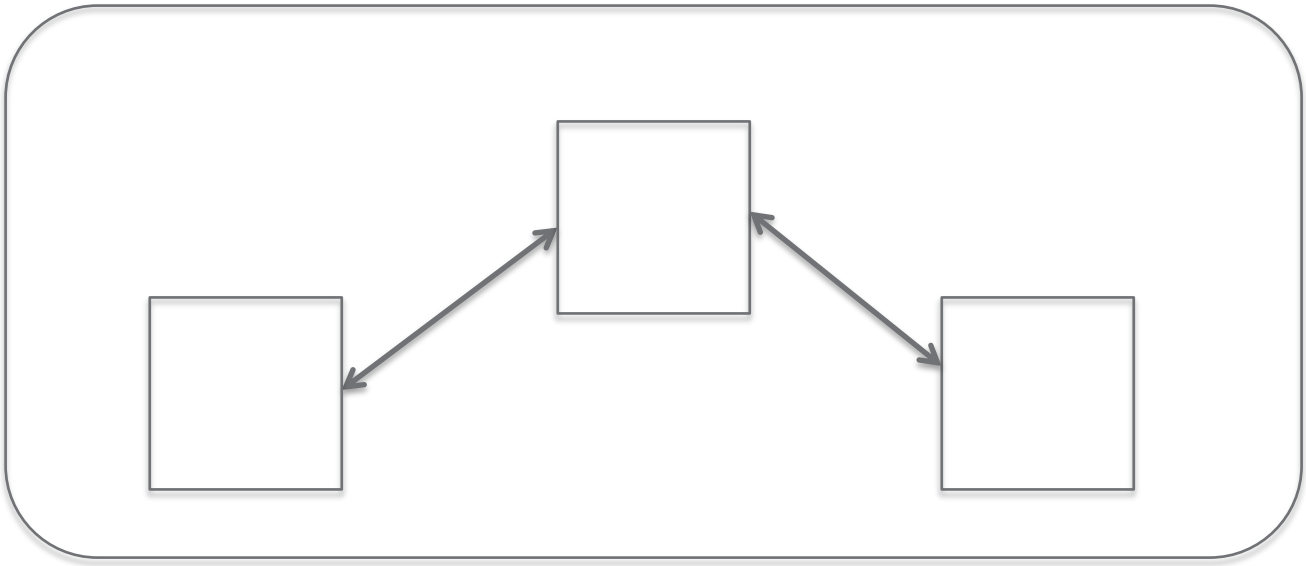

Figure 2. MS-SPEAK Translation Functionality

NRTC's existing WSR prototype hardware will be deployed inside a utility's network boundary. In a nutshell, pre-existing MultiSpeak platforms will be reconfigured to send data to the ESB+. The ESB+ will leverage WSR prototype software to scan the received network traffic (e.g., software such as "ngrep" or "justniffer"), then depending on configuration, the ESB+ will execute a predefined workflow to validate the received MultiSpeak, translate it to another specification, or both. The ESB+ will also be capable of executing functions against the received MultiSpeak (e.g., "hot socket" detection and notification). Two broad areas of work in the area of the WSR have been identified:

1. Production improvements to the WRS prototype
   (e.g., high availability to support SLAs, fault-tolerance, etc.)
2. Additional value-added functions
   (e.g. processing specific outage notification events)

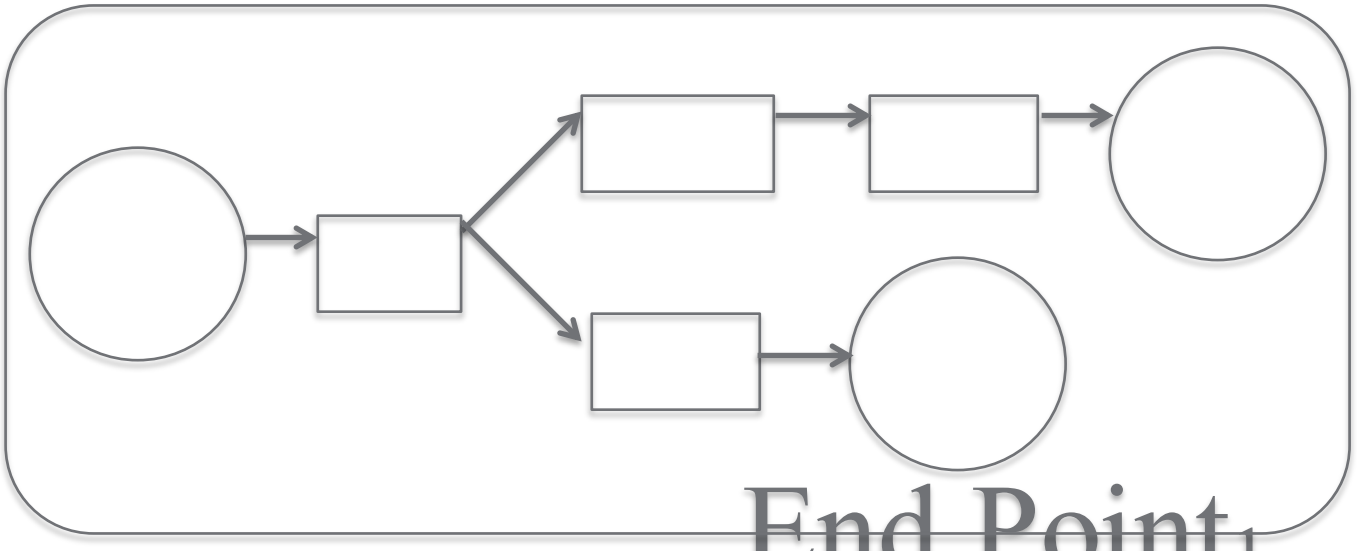

Figure 3. Example Workflow for Parallel Processing (Production and Testing)

2.2 High Level Workflow

Workflows may be created for any MultiSpeak message type using one or more of the actions below. See Figure 3 for a simple example of how the ESB+ could be configured to allow parallel processing of production MultiSpeak v3 messages for both transformations to v5 for testing and a pass-through of v3 for production continuity.

2.3 Workflow Outline

1. Network packet arrives at WSR
2. WSR determines if packet contains MultiSpeak
    a. If not, the packet is forwarded to a preconfigured destination
3. WSR determines the MultiSpeak message type
(i.e. server + method + parameter signature)
4. WSR searches a database to see if a rule has been defined for that message type, possible actions are:
    a. Forward
    b. Validate
    c. Translate
    d. Drop
    e. Log
5. Forward:
    a. The message is simply forwarded "as is" to a preconfigured destination
6. Validate:
    a. The message is validated using the following criteria:
        i. Well-formed XML
        ii. Existing MultiSpeak specification XSD
        (i.e. XML schema definition; data types, required fields, etc.)
        iii. A research opportunity exists to validate the content (i.e. data between XML tags) [N.b. The <comment> tag is an expedient integration tag that may contain numerous data types, which may invalidate the MultiSpeak specification and make interoperability brittle.]
7. Translate:
    a. If a known translation has been defined for the message type:
        i. WSR translates the message type according to the predefined rules (e.g., regular expressions) in the database
    b. If no translation exists, WSR is preconfigured to take one of the following actions:
        i. Forward
        ii. Drop

iii. Log

1. In learning mode, the message type is flagged for manual translation and a rule is created to allow for step 7.a to be taken next time the message type is encountered.

8. Drop
   a. The message is simply dropped with no further action taking place
9. Log
   a. Logging takes place according to both general, and specific configurations:
      i. Log all messages, with the option of:
         1. Full packet capture
         2. Meta data (source, destination, message type, etc.)
      ii. Log individual messages
      iii. Log ESB+ workflow at typical levels, including; DEBUG, INFO, WARN, ERROR.

2.4 MultiSpeak Message Translation

The scope of translating from one MultiSpeak specification to another is currently unknown. Entire end points in one version of the specification may not exist in another version of the specification. End points that are common to two versions of the specification may not share common methods. Even with common end points and methods, there may be differences in the number or data type of the method parameters.

One of our first tasks will be developing software that can enumerate the mapping between any two MultiSpeak specifications by reading the associated WSDL (web service discovery language) files associated with each specification. The concatenation of a MultiSpeak server and method will form a message type.

The method name, along with the parameter signature (i.e. number and type of parameters) will attempt to be found within a translation database. Six potential mapping cases between specifications have been identified in Table 1.

Recursion must be reduced to the base case (if possible). Solving the base case is an engineering problem to be solved by regular expression. In the naïve case of one-to-one mapping, translation may still be required due to a mismatch in the methods' parameter signatures. For example, both specifications may implement the same server [e.g., MR (Meter Read)] and the same function name (e.g., ReadMeter) with the same number of input parameters, but different data types for those input parameters (e.g. MeterNumber may be a String data type, and MeterNumber may be a complex data type, MeterNumber, in the other specification). Ultimately, complex data types map to simple data types, such as MeterNumber -> String. In these cases, automatic translation may be possible for message types that map 1:1 between specifications.

Message types that cannot be automatically translated must have translation rules written for them. The goal of our translation software is to abstract as much of the technical details regarding this

translation as possible from the end user. We expect the definition of a translation to be possible using a mouse (drag-and-drop) with keyboard shortcuts for accessibility. The end user will not need to have any knowledge of programming, regular expressions, or data types.

Message translations will always be 1:1. Phase 1 of MS-SPEAK will not contain any complex translations that implement translation in a many:1 or 1:many relationship.

2.5 Cryptography

Newer specifications for MS-SPEAK (i.e. > v3) allow for security in the form of Secure Socket Layer (SSL) / Transport Layer Security (TLS) which provides negotiation, authentication and encrypted communications. ESB+ will provide the capability to connect and communicate with network devices through SSL/TLS.

The software development described in this section covers the encryption services for receiving secure communications and is referred to as the *Crypto Listener*.

This software will be transparent to all other services described in the Enterprise Service Bus (ESB) specification. The output of this module will be decrypted XML as transmitted by the original sender. This software will not be affected by any trailing data stream requirements (e.g., translation, validation, forwarding). We will use well-supported third party libraries to handle encryption and manage certificate. It is beneficial to use an encryption solution that has been thoroughly vetted by the security community. Selection of the encryption libraries will be deferred until PNNL can perform additional analysis on encryption libraries and select one that meets the needs of the MS-SPEAK project. Factors such as programming language and platform support, outstanding defects, and support for the latest protocols will be important considerations in the ultimate selection.

The *Crypto Listener,* as shown in Figure 4, only covers encrypted communications. Standard v3 communications and non-secure later protocols are described elsewhere. This listener will run on a port separate from the non-secure listener and will not interfere with its operation.

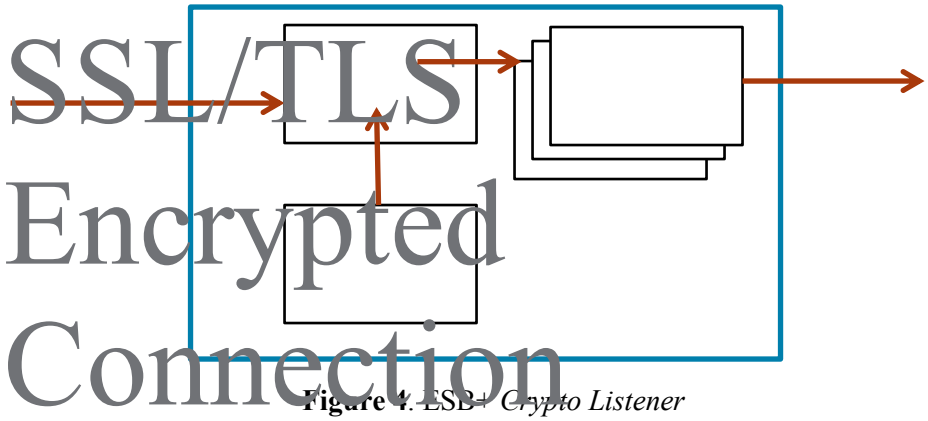

Figure 4. ESB+ *Crypto Listener*

The *Crypto Listener* will provide the following services:

- Listens on user selected port
- Certificate store with instructions on how to install new certificates
- Pass through decrypted XML to ESB+ Services
- Negotiation for appropriate encryption protocols
- Handle standards for certificate passing and authentication
- Allow multiple single-connection sessions
- Interface to the ESB+ logging/reporting capability for troubleshooting

2.6 Phased Approach

The *Crypto Listener* handling encryption will be provided in Phase I of MS-SPEAK. This will allow early testing with existing systems. Including this capability in the initial phase will allow secure integration early on to help characterize existing systems and how they handle trust. The early implementation will also encourage security in new products and installations since the ESB+ will be designed from the ground up to provide the latest in encryption standards. We will include the latest commercially supported version of TLS and SSL with backward compatibility to algorithms we identify in legacy equipment during integration.

The ability to encrypt and forward to the next listener will be provided in phase II if needed. The decision to defer this capability is driven by risk. When the ESB is running inside the utility protected network boundary the security risk is significantly reduced.

2.6.1 Out of Scope

Because certificates provide trust and identity management we will document how to install and use certificates with this software. We will not include any registered certificates with this development. Acquiring certificates requires registering as an individual, company or organization with a trusted root certification authority. If encryption is required the final user of this software should register and provide all identity information, or ask vendors to provide the appropriate authentication.

2.7 High Availability

Because MS-SPEAK will be deployed in a critical infrastructure environment, high availability is a design goal. High availability may also require a high level of software quality assurance.

3.0 Technical Approach

Sylus Studio has a graphical-based pipeline that uses the JRE v1.8 and Tomcat. Stylus Studio costs $250 per license and a 21% of total licenses / year maintenance fee. Source files are XSD and the target is XML. The first value add our team was able to develop was software that can automatically generate the required input XSD for the target using an existing MultiSpeak specification. Because this process is automated, we can generate the required XSD for any previous, current, or future MultiSpeak specification.

3.1 Java Pipeline

By deploying Stylus Studio's Java Pipeline to the Cloud, we will be able to implement custom workflows using multiple XSLT files. This solution allows for the creation of Java pipeline workflow services that need not be repeatedly purchased, installed, configured, and maintained multiple times by each utility unless they desire to control their own pipeline.

3.2 XSD Validation

XSD validation will be the first step in the pipeline. This is syntax validation. The pipeline will log any errors and alert the user. These errors can be corrected one of two ways. First, the sending process can be corrected. If it is not feasible to correct the sending process, MS-SPEAK can be used to translate the invalid XML to valid XML.

3.3 XSLT Transformation

XSLT translation is the core capability of MS-SPEAK. We began by translating the SOAP header, which is a key component of every single MultiSpeak message and is the same for a given specification. Thus only one translation algorithm is necessary to convert any MultiSpeak v3 SOAP header to a MultiSpeak v5 SOAP header. Focusing on the SOAP header gave us the necessary familiarity with the Stylus Studio tool and associated XML technologies to tackle the more complicated individual MultiSpeak message translations. Some cases (e.g., the base case) are a simple 1:1 translation. However, most messages (e.g., the edge cases) are much more complicated.

3.4 Translation Base Case

The ideal translation situation is when the end point exists in both specifications, and the function is also the same (i.e. same name and parameter signature). We refer to this case as the base case. The only required changes in the base case are syntactical and occur at the machine-to-machine level in the various protocols outlined in the Introduction section.

3.5 Translation Edge Cases

It is more likely that in addition to the syntactical changes required by the machine-to-machine protocols, MultiSpeak message elements and attributes will likely change from specification to

specification. In the worst case, a function that exists in one specification may not exist in another specification. Originally, we had hoped our translation function would attempt to translate an incoming message as if it were the base case. If this were not true, the workflow would log an error and alert the user that the message translation was an edge case, requiring user action to teach the workflow how to accomplish the translation.

The Stylus Studio Java Pipeline uses a single XSLT to point to additional XSLTs that will describe how to translate a specific MultiSpeak message from one specification to another. Often the software tool can attempt to complete this translation using an automapping tool. Because the resulting XSLT is machine generated, it will need to be checked by a SME (subject matter expert) to ensure the translation is correct.

Occasionally, there will be edge cases that do not directly map from one MultiSpeak specification to another. For example, v4.x added AVL (Automatic Vehicle Location) messages. There is no corresponding v3.0 message. In the case of translating from v3.0 to v4.x, this should not cause a problem because no input messages exist. However, translating from v4.x to v3.0 would be problematic because there is no mapping from the message to the old specification. In this case, the Java Pipeline will have to be instructed how to handle the message; some options include; forward the message as-is (likely to break the v3.0 system), drop the message (likely to cause process problems), or log the message and alert the user. It may be possible to provide translation for some messages by separating required elements and transmitting multiple messages to multiple end points, but that is outside the scope of our Phase 1 development efforts. This year we are focusing on edge cases that still result in a 1:1 translation, such as:

- The same MultiSpeak message exists in both specifications, but each occurs in a different end point.
- Equivalent MultiSpeak message exists in both specifications in the same end points, but the name of the function is different.
- The same MultiSpeak message exists in both specifications in the same end point with a different parameter signature (may be the same variables in different order, with different data types, or different variables altogether).
- The same MultiSpeak message exists in both specifications in the same end point, but the XML element composition is different.
- Lastly, some combination of multiple edge cases may also occur.

As long as the resulting edge case is a 1:1 translation from one MultiSpeak message to another MultiSpeak message, we can create an XSLT that the Java Pipeline can use to perform the translation.

3.6 Automapping

Our analysis of the v3 and v5 standards of MultiSpeak revealed that an automated machine translation between standards is not feasible. MultiSpeak v5 is not backwards compatible with v3 because it not only renames and moves many functions, but also implements many new functions. Stylus Studio's Automapping feature is capable of aligning some elements from both the v3 and v5 WSDL files, but full mapping requires domain knowledge from a subject matter expert in industry to complete the mapping. See Section 4.0 Future Work for more information on how we envision this mapping to be accomplished in the future.

3.6.1 Step-by-Step Instructions to Translate between MultiSpeak Standards

The following steps explain how to complete a mapping between any two versions of MultiSpeak to enable translation via XSLT in MS-SPEAK:

1. Make an XSLT file to translate the MultiSpeak Request using Stylus Studio
   (Tools, Options, XSLT Editor: Default XSLT Version = 2.0
   Tools, Options, XSLT Editor, Processor Settings = Saxon)

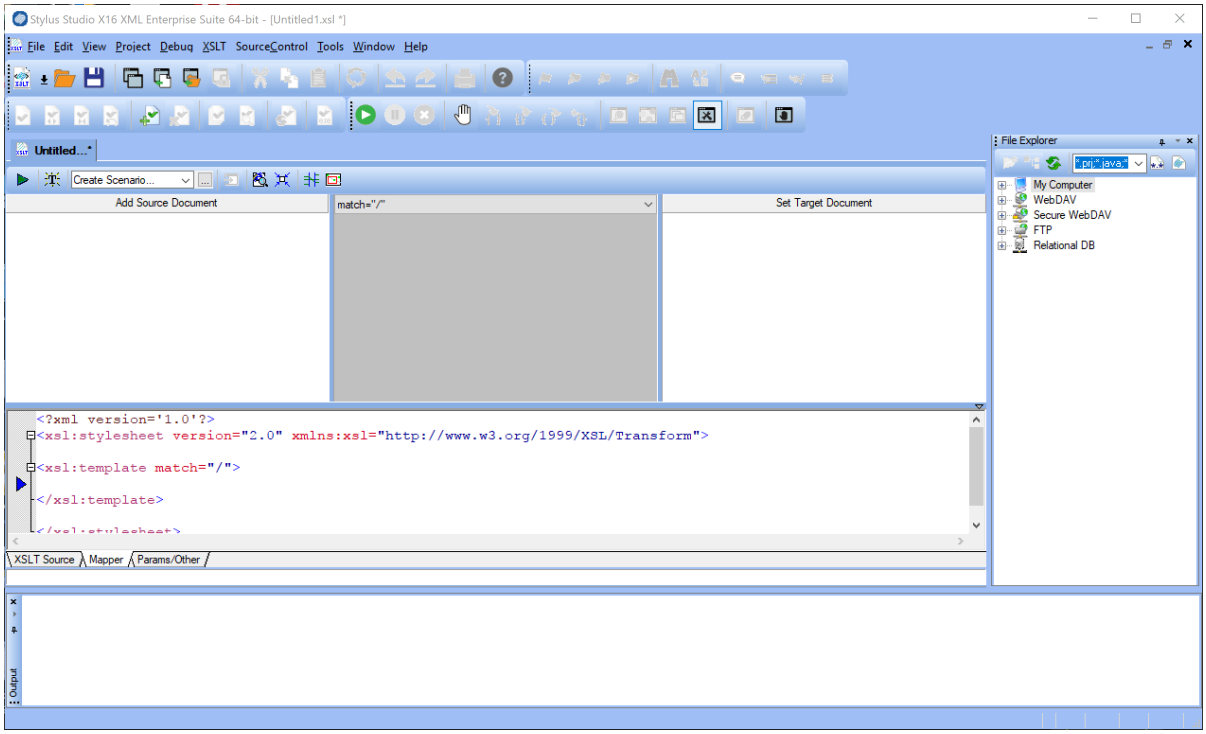

Figure 5. Stylus Studio

a. MDM_Server.wsdl (v3)

i. ODEventNotification

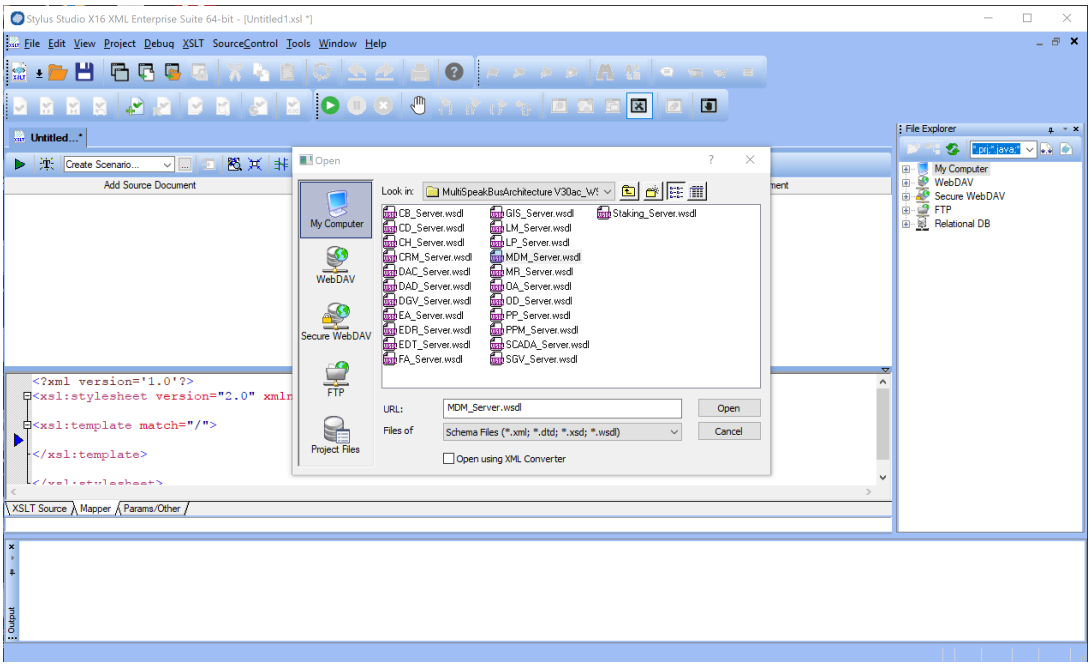

Figure 6. Opening the v3 MDM WSDL

b. NOT_Server.wsdl (v5.04)

i. ODEventNotification

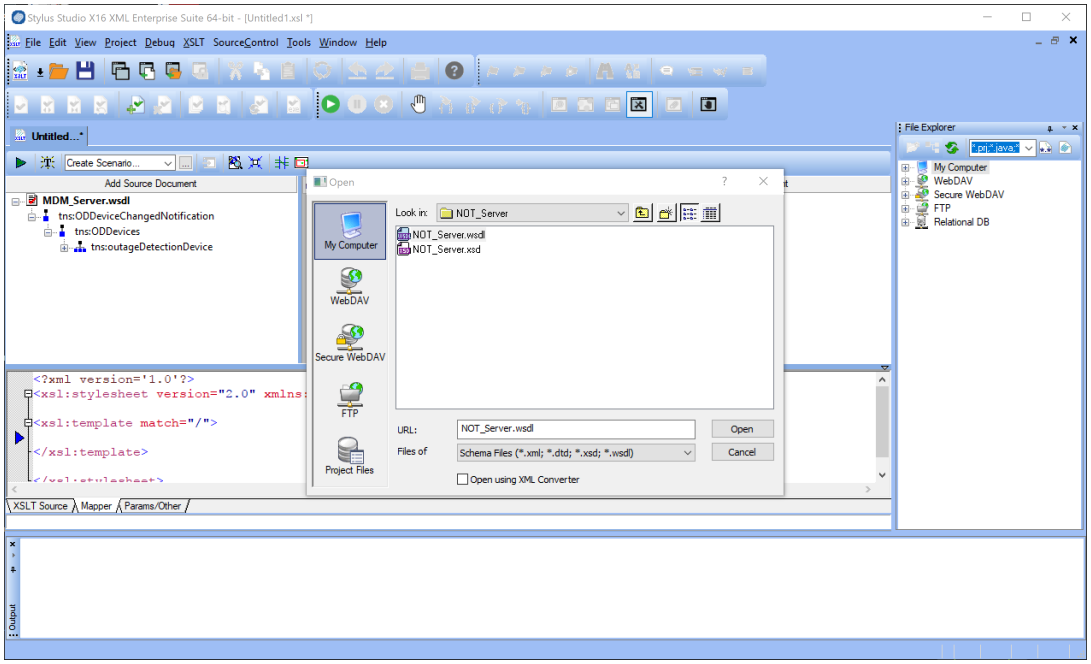

Figure 7. Opening the v5 NOT WSDL

c. Autolink bases in Mapper

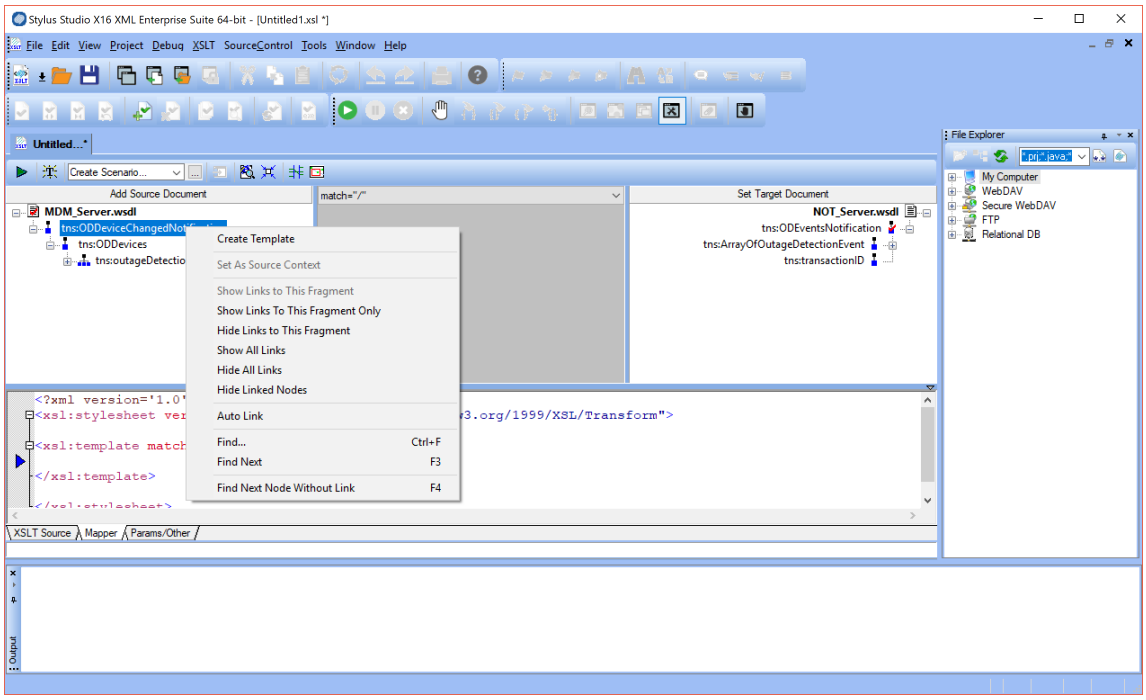

Figure 8. Autolinking the Bases

d. Autolink event in Mapper

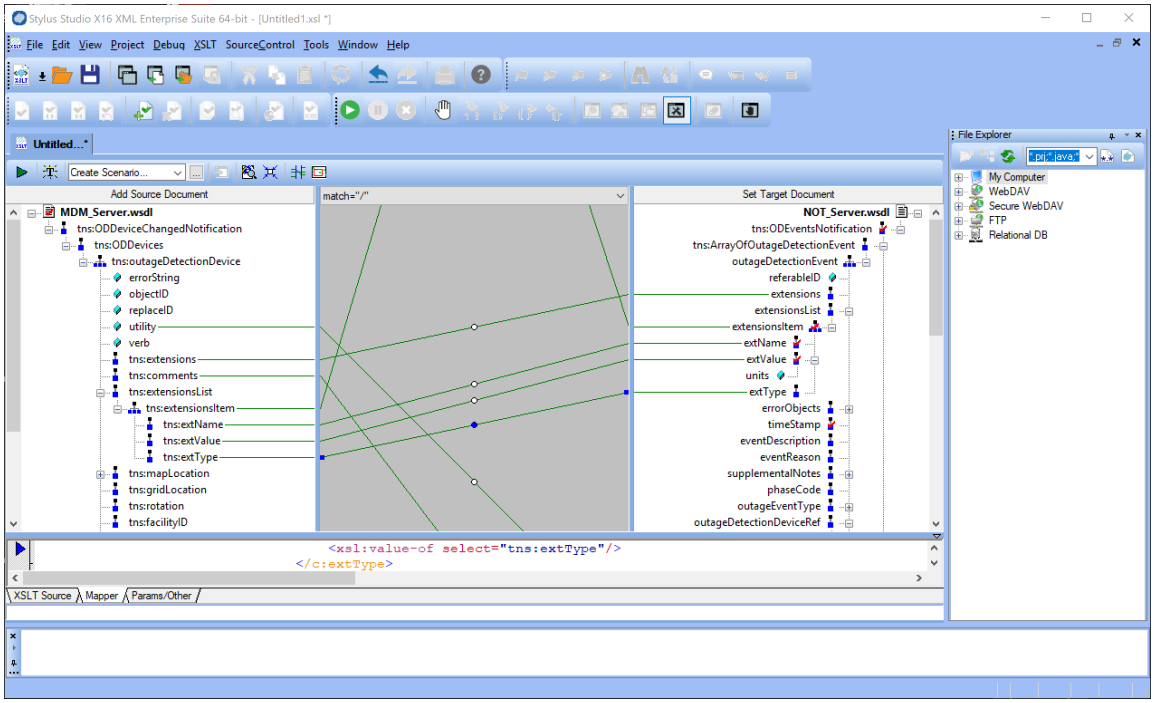

Figure 9. Results of the Autolinker

e. Save resulting XSLT file to:
(./Pipeline/JabbaPipe/xslt/eps)

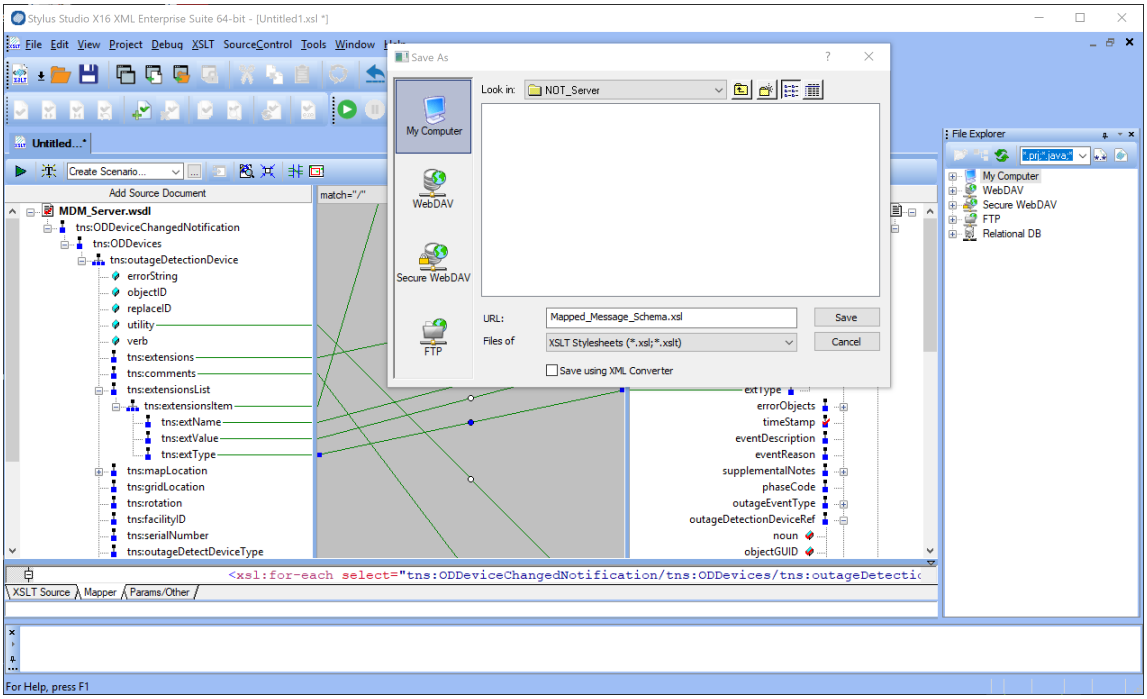

Figure 10. Save the Resulting XSLT

2. Import:
<xsltimport href=[XSLT file from #1] /> # line 12
3. Link XSLT in ./Popeline/xslt/lib/lib-msgheader-pnnl.gov.xslt
4. Edit XSLT
   a. One namespace per line for readability # line 2
   b. Edit the match tag:
      i. match = “tns:ODEventNotification”
   c. Remove first line
   <?xml …
5. Start three consoles on the cloud server:
   a. Console 1:
      i. cd ./Pipeline/JabbaPipe
      ii. . bld.sh
      iii. java JabbaPipe -d 1 -lc –e
   b. Console 2:
      i. cd ./Pipeline/Jitm

ii. python v3Client.py -i ODEventNotification-RQST.xml -L
(N.b. Remove first three lines if generated with MultiSpeaker:

1. REQUEST
2. MDM
3. SoapAction: [url]

c. Console 3

i. python v5Server.py –i ODEventNOtification-RSPN.xml

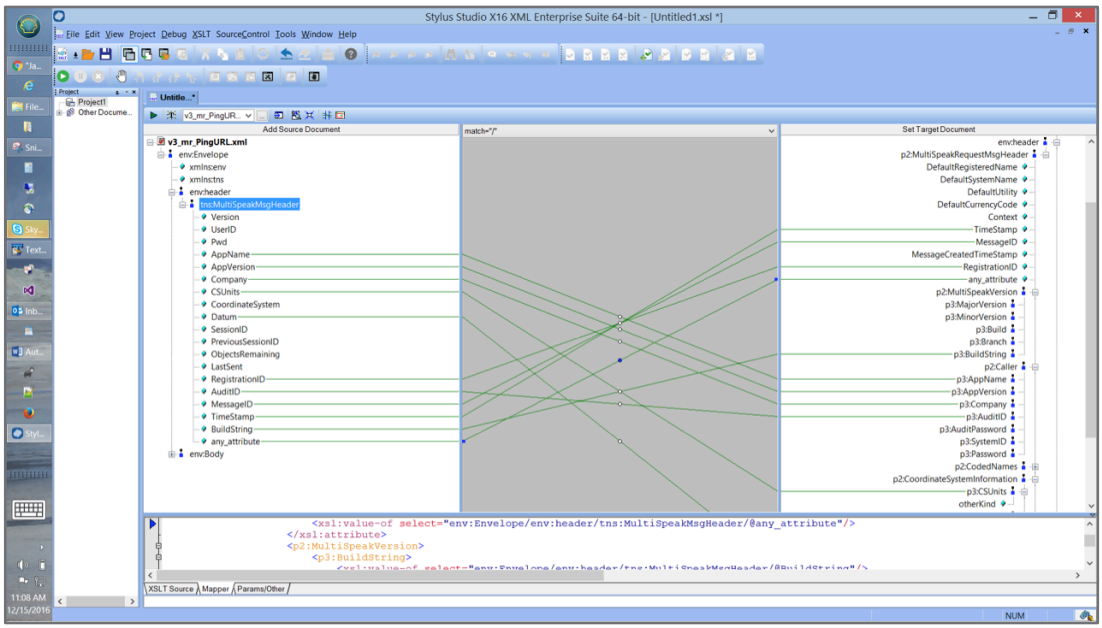

Figure 11. Stylus Studio Automapping Feature

3.7 SME Input

As you can see in Fig. 5, the automapping feature was not able to map each element from the XSD on the left (MultiSpeak v3.0) to the XSD on the right (MultiSpeak v5). In most cases, our development team will need assistance from an SME to explain how each element should map from one specification to another. Once an SME has completed the mapping, the resulting XSLT can be used by any utility to translate any number of corresponding MultiSpeak messages from one specification to another.

A community of interest that focuses on the creation, sharing, and modification of these XSLTs would significantly reduce the redundant work required by utilities to translate their own MultiSpeak messages.

3.8 Logging

State awareness of the workflow pipeline is critical to implementing the initial translation functions (i.e. XSLT files), alerting the user to messages that cannot be translated (and thus require a new XSLT), and assurance that incoming messages are being translated properly.

Logging failed validation of the input specification and failed translation (the resulting XML fails to validate against the output specification XSD) and alerting the user to these log entries was a critical development effort. The MS-SPEAK team is using logging to evaluate our workflow, which so far has been based on synthetically generated MultiSpeak traffic using *MultiSpeaker*, focusing on translation from the v3.0 specification to v5.

4.0 Future Work

PNNL has developed and deployed an end-to-end prototype that demonstrates how to perform translation, and benefits (validation, logging, translation). PNNL alone lacks SME knowledge to generate the necessary XSLTs and complete the required mappings to enable large scale translations between MultiSpeak standards. By choosing a single industry partner, we risk overfitting the translations to a single use case with that industry partner. This solution will not generalize well to the entire industry. By involving multiple industry partners, we risk length delays with a committee approach to translating hundreds to thousands of functions.

The creation and stewardship of a community of interest focused around creating, sharing, and modifying the requisite XSLT documents to increase the usability of MS-SPEAK, and therefore enable interoperability between MultiSpeak standards using MS-SPEAK will be of the utmost importance.

4.1 XSLT for Additional MultiSpeak Functions

By replaying full packet capture network traffic from a utility; MS-SPEAK will generate log output for any message that fails to translate. This log output will be informative to the MS-SPEAK team as to which MultiSpeak messages to create XSTL files for next. As we learned from prior work with the MultiSpeak protocol, messages are often infrequent and generally a small subset of the specification. By observing real network traffic, and working with an IAB (Industry Advisory Board) and SMEs, the MS-SPEAK team can focus on creating XSLT files that will have the most impact on interoperability between different specifications of MultiSpeak.

5.0 MultiSpeak "Secure Enterprise Access Kit" MS-SPEAK Test Plan

5.1 Overview

The NRTC and its identified contractors have been commissioned by PNNL as a testing agent of the PNNL developed MS-SPEAK functional framework. Two phases of testing will provide a sufficient determination of a DOE-defined "go/no-go" decision during the 3-year performance period. NRTC plans to provide a *static* test of the framework to exercise the fundamental components of the buss architecture. Following this initial test, a *dynamic* test will be executed in the field where demonstrated and witnessed by a program advising rural electric cooperative. NRTC will also provide subject matter expertise regarding MultiSpeak and utility operations supporting PNNL staff assurances of appropriate verification and validation of the MS-SPEAK functional framework.

5.2 Test Objectives

NRTC will provide test steps to support the following objectives:

1. Execute a verification of the phase-I prototype codebase.
2. Execute a validation of selected protocol endpoint functions.
3. Coordinate on-site field-testing.

As part of the verification objective, NRTC will demonstrate in its development environment a *static test* of the end-to-end software design. This verification test will verify the functional modules that have been integrated into deployable services.

1. Demonstrate the fully functional end-to-end MS-SPEAK capability on one or more methods as prototyped by PNNL's development team.

Next, a validation object to demonstrate a fully implemented MultiSpeak protocol functional assertion will be demonstrated in an on-site *dynamic test*.

2. Demonstrate the functionality one or more methods in an active field test leveraging an NRECA Electric Cooperative member that currently uses MultiSpeak in one or more commercial vendor implementations within their operational framework.

The MS-SPEAK verification and validation objectives will be sufficient to allow PNNL a 3rd party assessment of the capability, provide assurances that the work to date supports relevant field operations, and gain adoption support by an electric cooperative partner that has implemented the protocol in a progressive and effective manner.

5.3 Static Test

Performed with the MultiSpeak v3 MR:PingURL method, generated with *Multispeaker* synthetic traffic generator.

5.3.1 Requirements

1. Witness a functional framework demonstration of the YR-1 development effort.
2. Provide feedback to PNNL in support of, or critique of the outcomes of the initial demonstration.
3. Verify that the selected MultiSpeak endpoint method adheres to the version specification.
4. Verify that the framework can implement a selected version dynamic conformance of the protocol method.

5.4 Dynamic Test

Performed with the MultiSpeak v3 MDM:ODEventNotification method, forwarded from an NRECA utility.

5.4.1 Requirements

1. Deploy the MS-SPEAK codebase on an active electric cooperative partner network.
2. Test and report the passive message ingest functionality on a selected MultiSpeak endpoint method.
3. Test and report the method(s) being tested that the ingest is successful through observations and capture of operational data that is flowing into the passive MS-SPEAK bus listener.
4. Assert a version translation from the electric cooperative's native operational data to a selected version (newer or older) through observations and capture of this data stream.

5.5 Test Environment

5.5.1 High-Level System Architecture

The figure below (Ref. PNNL "GMLC 00068 MS-SPEAK Technical Requirements.docx") depicts a single MultiSpeak assertion.

A MultiSpeak "assertion" is a formal term used in defining an implementation of any combination of endpoint function(s). Early adopters of the protocol were required to submit such assertions to the formal MultiSpeak governing technical committee because there were no early version reference models for developers to follow. As the specification was rapidly being adopted and commensurate changes to subsequent versions being advanced, these assertions created a pseudo control over impending conformance desires by the technical committee.

Figure 12. Existing MultiSpeak Functionality

End

Figure 13. MS-SPEAK Functionality

Figure-1's endpoint depiction is a simple point-to-point assertion that will be used in both verification and validation stages.

The next illustration depicts the fundamental principle of the MS-SPEAK translation capability.

5.5.2 Test Bench

A fully functional test bench has been constructed and used extensively in the test and evaluation of correct protocol use, correct implementation methods, and full fidelity field readiness testing. The modules are depicted in Fig. 3. This bench replicates a utility framework where we are able to generate end-to-end data streams of MultiSpeak endpoint assertion models. The "L2relay" function contains the code necessary to *capture*, *parse*, *process*, and *post* MultiSpeak messages across a high-fidelity network. We have also implemented a virtual test bench in simple tools such as VBox, as well as fully functional cloud-based (Amazon Web Services) environments. The goal of the test bench frameworks is to demonstrate the modularity and diversity of the MS-SPEAK processing methodologies. There may be cases, as advised by our utility partners, that are vendor-hosted environments for example, and the MultiSpeak AMI endpoint data is brokered by the Vendor to the utility for use. In This case, the MS-SPEAK platform would act as the mediator, providing initial conformance testing, routing, and security management of the vendor supplied connection.

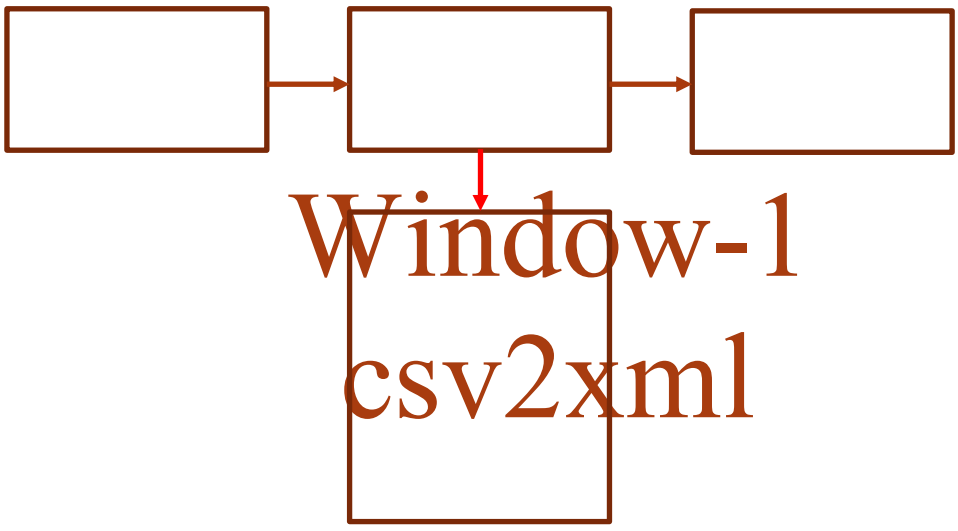

Figure 14. MS-SPEAK Test Bench

5.5.2.1 System Description

1. Normal operation requires the following python scripts.
   a. **oms-http.py** (Simulated outage management system and message responder for MultiSpeak endpoint destine data)
   b. **ms-sniffer.py** (the core 'sensor' for passively listening for MultiSpeak payloads)
   c. **l2relay.py**, or **substa-l2relay.py** (the relay for creating L2 bound JSON (JavaScript Object Notation) payloads that have been extracted by the ms-sniffer)
   *NOTE: There are two "relays" from L1 to L2 as currently we're developing in an agile context. The **l2relay.py** supports the meter data, and the **substa-l2relay.py** for substation readings, both of which support their respective csv formatted data. Future code will be a single daemon and discriminate on a source identifier.*
   d. **csv2xml.py** (the initiator which takes a formatted csv data file, ingests it, encodes the data into protocol structures, and sends it with full Ethernet fidelity to the oms-http server)
2. Each module has an associated 'script' named the same as the source code described above but without the .py extension. The python runtime environment is triggered by a command ./{scriptfilename}, in our case each script will be fired off manually such as "./oms-http" for the oms server.

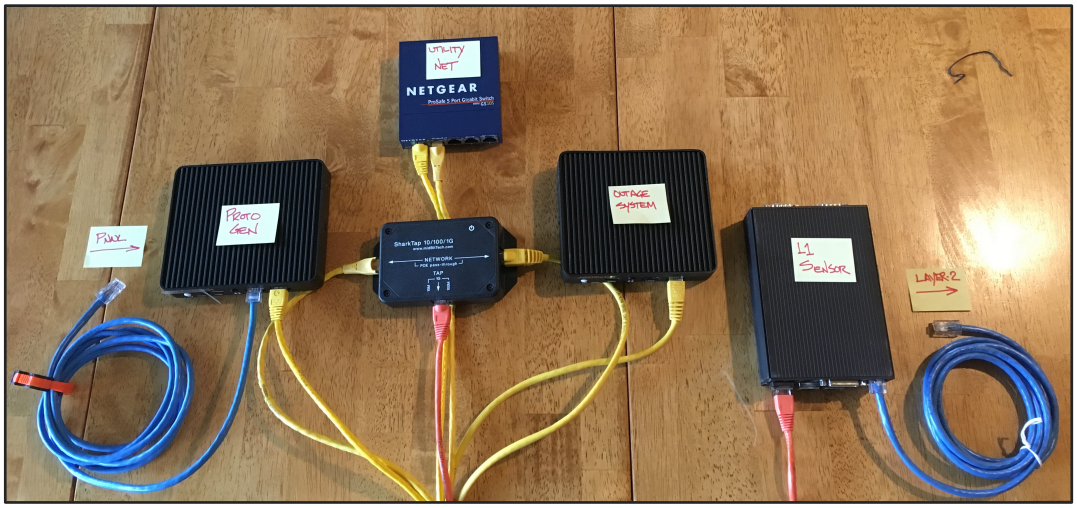

Figure 15. MS-SPEAK Physical Test Bench

5.6 Static Test Procedure

Environment: BlackByte Cyber Security, LLC – Server: BBYTE4 (Dev/Test)

1. Login: as req'd
2. Test environment settings
   a. {execute "easy button"}
      i. required environment parameters
      list

list
list

3. {start jabba pipeline}
   a.
4. {execute stream}
   a.
5. {}

Observations:

1. MultiSpeak V.3 “PingURL” message *in*
   a. Code snippet
2. MultiSpeak V.5 “PingURL” message *out*
   a. JABBA-Pipeline stdout
   b. *Code snippet*

Conformance:

1. Protocol acceptance (MS-SPEAK)
2. Protocol rejection (MS-SPEAK)

5.6.1 Static Testing Results

Static Testing Observation Checklist

X	Network packet arrives at WSR
X	WSR determines if packet contains MultiSpeak
	If not, packet is forwarded to preconfigured destination
	WSR searches a database to see if a rule exists for that message type, possible actions are: *Forward (as is), Validate, Translate, Drop, Log*
	Forward: The message is simply forwarded "as is" to a preconfigured destination
X	**Validate:** The message is validated using the following criteria:
X	Well-formed XML
X	Validates using existing specification XSD (XML Schema Definition)
X	**Translate:** *If a known translation exists for the message type: WSR translates the message type according to the stored rules and regular expressions in the database*
	If not, WSR chooses another preconfigured action: *Forward (as is), Drop, Log*
	*The message type is flagged for manual translation
/	**Drop:** The message is simply dropped with no further action taking place
/	**Log:** *Logging takes place according to both general, and specific configurations:*
	Log all messages, with the option of:
	Full packet capture
	Meta data (source, destination, message type, etc.)
/	Log individual messages

X = Successfully observed
/ = Partially observed

5.7 Dynamic Test Procedure

5.7.1 Dynamic Field Testing Results

Dynamic Testing Observation Checklist

X	Network packet arrives at WSR
X	WSR determines if packet contains MultiSpeak
	If not, packet is forwarded to preconfigured destination
	WSR searches a database to see if a rule exists for that message type, possible actions are: *Forward (as is), Validate, Translate, Drop, Log*
	Forward: The message is simply forwarded "as is" to a preconfigured destination
X	**Validate**: The message is validated using the following criteria:
X	Well-formed XML
X	Validates using existing specification XSD (XML Schema Definition)
/	**Translate**: *If a known translation exists for the message type: WSR translates the message type according to the stored rules and regular expressions in the database*
	If not, WSR chooses another preconfigured action: *Forward (as is), Drop, Log*
	*The message type is flagged for manual translation
/	**Drop**: The message is simply dropped with no further action taking place
/	**Log**: *Logging takes place according to both general, and specific configurations:*
	Log all messages, with the option of:
	Full packet capture
	Meta data (source, destination, message type, etc.)
/	Log individual messages

X = Successfully observed
/ = Partially observed

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

U.S. DEPARTMENT OF
ENERGY

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7665)

www.pnnl.gov