Developing Data Converter and Validator for Air Traffic Management System

Jungwon Zang
Master of Software Engineering
Embry Riddle Aeronautical Univerisity
Nextgen ERAU Applied Research (NEAR) lab
Daytona Beach, FL
zangj@my.erau.edu

Abstract— With the rapid growth of global air travel, there is an urgent need for reliable air traffic management system. Over the past ten years, Federal Aviation Administration (FAA) has been involved in the development of a System Wide Information Management (SWIM) data exchange system that integrate vital aircraft, flight, air traffic, and weather data across the nation and more recently across the globe. In such a system, delivering the right information to the right stakeholders at the tight time is critical for the safe and efficient air travel. Currently, SWIM data exchange is facilitated through XML files, however some the data producers and the consumers of this data to the SWIM uses JSON for the transfer of their information. Java language was primarily used for XML to JSON data conversion. In order to guarantee the validation of this conversion, there are number of validators that have been implemented using Java language. However, the teams at NEAR lab recognized that Java may not be the most efficient implementation language to accomplish these tasks. The purpose of this research was to first identify potential alternative implementation language, which can be used for implementation of converter and validator modules. After validation and verification of alternative implementation, evaluate much more efficient the new language comparing to the current Java implementation.

Keywords—xml, json, javascript, typescript, validator, converte, SWIM, ATMr

I. INTRODUCTION

With the rapid growth of global air travel, it is important to keep the air traffic management system reliable. Over the past few years, FAA has been involved in the development of SWIM data exchange system. SWIM that stands for system wide information management is the digital data-sharing architecture for NextGen project. Air traffic management deals with flight, weather, and aeronautical data across the nation in XML format. Currently, SWIM data exchange is facilitated through XML files, however, there are some crucial issues with XML that affect the integrity and efficiency of data exchange. As XML has limitations, JSON has emerged in web services and the data stakeholders uses JSON for the transfer of their data. In SWIM system, Java language was primarily used for XML to JSON conversion and validation. The teams at NEAR lab recognized that other programming language might replace Java to improve efficiency for the implementation of converter and validator and this inquiry led me to research potential alternative language to accomplish conversion and validation.

II. RESEARCH OBJECTIVES

First, the research for building a converting XML to JSON has been conducted in the research lab at Embry Riddle Aeronautical University's NextGen ERAU Applied Research (NEAR) lab. The lab has developed a data converter with Java language. However, the lab abandoned using Java more than five years ago. The reason Java was abandoned is not so much technical problem as no one wanted to write or maintain Java code. At the time of rewriting the lab's dispatching tool, the team decided to exclusively use Node.js with TypeScript on both frontend and backend environment in order to minimize maintenance. Currently, the team is using an NPM module that depends on JAXB to convert the FIXM/FFICE XML Schema to JSON Schema with the *xml2json* package. Using JAXB in data converter, JAXB has less control on parsing than SAX or DOM. This happens because JAXB is a higher layer API. While converting XML to JSON files, JAXB has some overhead tasks which makes it much slower than something like a SAX. About JAXB in XML validator, one of the disadvantages is that it requires DTD (Document Type Definitions), so we cannot be able to use JAXB to process generic XML. As JAXB cannot process generic XML without DTD, this is going to be the main disadvantage of JAXB in XML schema validator. They have

started to research building an aviation data converter and XML schema validator with a new programming language to replace the old version of the converter to investigate the feasibility, usability, maintainability, and ease-of-use aspect of utilizing either language besides Java dealing with the XML dataset. JavaScript application catches errors early during coding and can make code management and debugs easier by static typing. In this research, I expected the data converter written in Node.js with JavaScript and XML schema validator with TypeScript.

Second, the air traffic management stakeholders highlight the necessity of the business rule validation of flight planning. In this research, I expect the business rule validation of the flight plan that checks and validates all required values and reasonable values present in the file with TypeScript.

III. RESEARCH RESOURCES

Data sharing in ATMs becomes an important component. The ATM system will increasingly be more reliant on accurate and timely information sharing. For a safe, efficient, and more predictable ATM system, the Federal Aviation Administration (FAA) led the Next Generation Air Transportation System (NextGen) [1]. In Fig 1, NextGen Information Sharing Architecture consists of four representative stages: NextGen Applications, Standards for Data Exchange, SWIM Messaging Infrastructure, and FTI IP Backbone stage. This research is primarily focusing on SWIM Messaging Infrastructure and Standards for Data Exchange. SWIM is an information-sharing framework that supports the point-to-point exchange method by publishing data and transmitting it to ATM data distribution systems based on Service Oriented Architecture (SOA) message patterns. It improves safety by receiving and delivering the right information at the right timing and efficiency by enabling performance-based operations and a service-oriented environment. SWIM communicates more flexibly and cost-effective and improves connectivity and interoperability for information exchanges.

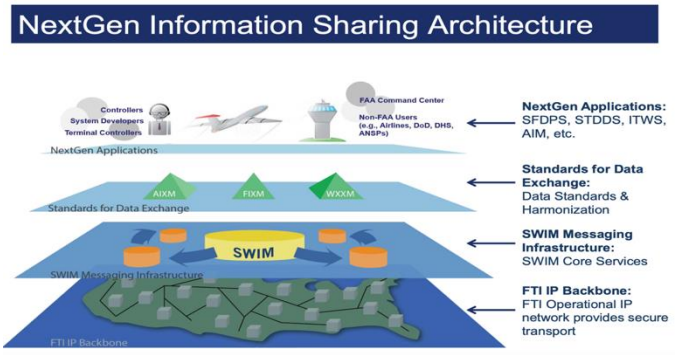

Fig 1. NexGen Information Sharing Architecture

Fig 2 is an overall SWIM architecture. SWIM provides the infrastructure, standards, and services to optimize the accurate and swift exchange of relevant data for National Airspace System (NAS) systems and the air traffic industry. SWIM aims for facilitating more efficient data sharing of ATM system information. There are three representative aviation datasets: Flight, Weather, and Aeronautical data. The FAA and the ICAO work to create the aviation versions of eXtensible Markup Language (XML) files that have become standards for the exchange of information over SWIM. All datasets have each global exchange standard information format: the Flight Information eXchange Model (FIXM), the Aeronautical Information eXchange Model (AIXM), and the Weather Information eXchange Model (iWXXM). SWIM is the model that uses SOA to facilitate the development of up-to-date systems and to harmonize that three-aviation information for all air traffic management stakeholders.

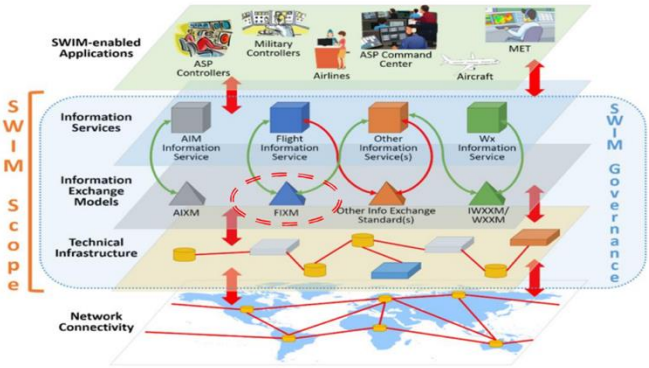

Fig 2. SWIM Overall Architecture

To achieve the research objective, highlighted in the previous section, there was a need to identify and leverage datasets that contain information about flight and weather.

1. AVIATION DATASETS

- FLIGHT INFORMATION

Flight data provides information and assists air traffic management for a safe and efficient flight. About flight information, it is comprised of two information sets: Flight and Flow information. FIXM which stands for Flight Information eXchange Model is a global data exchange model that captures Flight and Flow Information and harmonizes the flight information data structure of the scope of FF-ICE. FF-ICE which stands for Flight and Flow Information for a Collaborative Environment provides information for flow management, flight planning, and trajectory management with the ATM operational components. FLXM which stands for Flow Information eXchange Models is a new data standard by FAA for Air Traffic Flow Management (ATFM) information exchange to implement aviation information exchange standards. It provides flight-specific elements that control departure time, control arrival time, control route segment, and control speed [2].

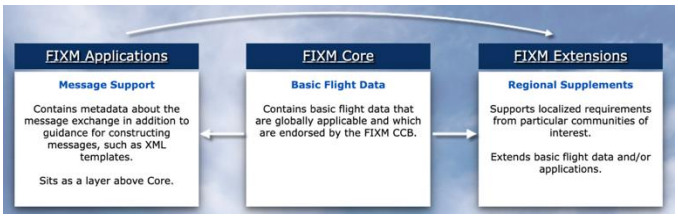

Fig 3. FIXM three architectures and relationship

- WEATHER INFORMATION

IWXXM (ICAO Meteorological Weather Information eXchange Model) reports aviation weather information in XML and GML format. IWXXM represents METAR, TAF (Terminal Aerodrome Forecast), SIGMET (Significant Meteorological Information), AIRMET (Airman's METeorological Information), Tropical Cyclone Advisory, Volcanic Ash Advisory, and Space Weather Advisory [3]. METAR reports the weather information to assist the weather forecasting by receiving weather report from aircraft pilots and meteorologists who aggregates METAR information. It contains information on the temperature, dew point, wind direction, and precipitation data. TAF reports the weather forecasting information by being issued four times a day from the center of an airport runway. SIGMET contains meteorological information on the safety of aircraft that consists of convective and non-convective types. AIRMET concisely describes the weather phenomena that occur along an air route and provides information about turbulence, surface winds, and visibility. Tropical Cyclone Advisory indicates the information that a tropical cyclone has formed and Volcanic Ash Advisory reports whenever a volcanic event occurs.

2. DATA EXCHANGE FORMATS

- XML

The eXtensible Markup Language (XML) is a web language for encoding documents in a way that is both human-readable and machine-readable, and it also provides strong support via Unicode for textual data. It is used to simplify the data sharing process and transfer data between hardware and systems or applications and for the representation of arbitrary data structures such as those used in web services. XML makes the data creation and transfer between different incompatible systems easily. The ultimate design goals of XML focus on simplicity, generality, and usability across the Internet. Compared to HyperText Markup Language (HTML), XML programmers can define their tags according to the application. In HTML, the programmer should change the data every time to display the data, however, the XML can store separated and can be read using JavaScript as an external XML file. Although XML has more benefits than HTML, XML is not the perfect replacement for HTML. Nowadays, JSON can be used as an alternative to XML because its syntax is easier to read and write.

- XSD

Document Type Definition (DTD) is precisely describing XML language and is used to define the structure of an XML file and contains a list of elements and validate the file. An XML Schema Definition (XSD) is based on and written on XML. It is recommended by W3C (World Wide Web Consortium) for replacing the DTD. Fig 5 is the basic example of an XSD file. XSD is defined to describe the structure, data types, and elements of the XML documents and to support namespaces. XSD is used to validate the structure and the vocabulary of an XML document against the grammatical rules of the appropriate XML language. For validation, XSD is created as a separate file, and it is linked with the respective XML document with specifying XML schema specification in the root of the XML document. An XML document should be well-formed and validated. The well-formed XML document means that the XML rules such as nesting the tags, opening, and closing tags correctly have been used. XSD is the best definition for defining the structure and the content of an XML document

- JSON

In recent years, JavaScript Object Notation (JSON) has become an open standard format for text data interchange. Using this format, data is transmitted to web services in a computer-readable and computer-understandable way. Data is mostly transferred between systems using JSON, that is, serialized data is transmitted through a network connection very fast and in a simple way between a server and a web application. JSON is typically much easier and fits nicely into a most programming languages. According to serializing and transmitting data, JSON is faster since it is designed specifically for data interchange and transmission time is lower because JSON does not need a namespace. Along with JSON development, in air traffic management, stakeholder want to use JSON's easy-to-understand structure than complex XML structure. This is my first research objective of data converter.

IV. CONVERSION METHODOLOGY

1. SYSTEM ARCHITECTURE

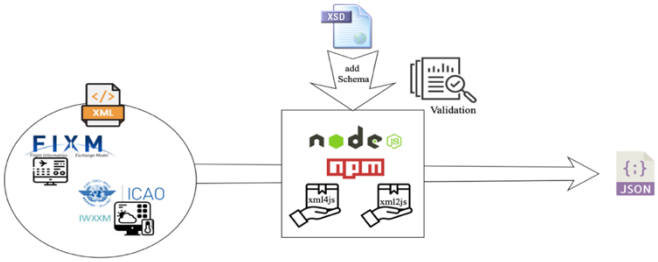

Fig 4. Data Converter with Validator Architecture

Fig 4 is the overall architecture of the system. For the input XML files, the project dealt with only flight and weather datasets, which are called FIXM, FF-ICE and FLXM, and IWXXM. The converter system is designed with Node.js and NPM packages both *xml4js* and *xml2js* and the validator file is designed using XML Schema files for validating whether the XML file is well-defined and XPath for extracting information from parts of XML document such as an element or attribute. The outcome should be a JSON file that converts the input XML file of FIXM, FF-ICE and FLXM, and IWXXM.

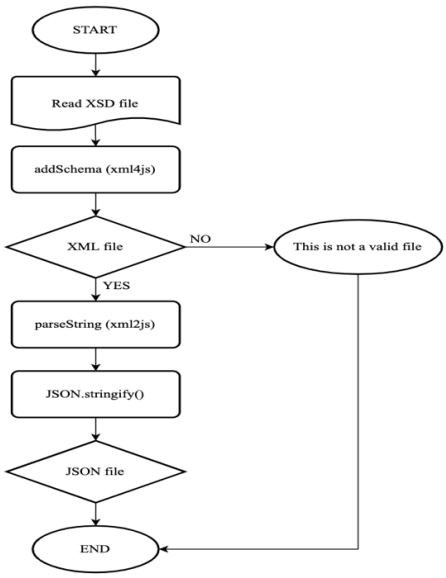

Fig 5. Flow chart of the converter with XML Validator

Fig5 is the flow chart of the data converter with the validator. It starts by reading the XSD file and finding all schemas that are presented in *xs: import* and *xs: include*. Then, the addSchema() function of *xml4js* package is adding all found schemas to the system and checks the validation between the XML file and XSD file with matching schemas. If the validator finds the schema is not matching between the XML file and XSD file, it prints out "This is not a valid file" and the system directly ends. Each step is described in detail with a flowchart or diagram in the following sub-sections.

2. BUILDING A TOOLKIT

◆ NODE.JS

The converter design includes Node.js and NPM modules. Node.js [4] is an open-source server environment that runs JavaScript on the server. JavaScript code can only be run using the V8 Engine, which is supported by all modern browsers. The Node.js platform allows JavaScript to be used for command-line tools as well as server-side scripting, which produces dynamic web page content before it is sent to the user's web browser. Consequently, it represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js is an event-driven architecture that performs asynchronous I/O. These choices aim to design for optimizing throughput and scalability in web applications with many input and output operations, as well as for real-time web applications. It stands out in real-time web applications performing push technology over web sockets and has web applications with real-time and two-way connections where both the client and server can communicate by allowing them to exchange data freely. The main point of Node.js is remaining efficient in terms of data-intensive real-time applications that run across distributed devices with non-blocking and event-driven I/O.Node.js has a set of built-in modules that do not require any further installation.

◆ NPM MODULES

NPM stands for Node Package Manager. NPM is the online repository and package that contains all the files for a module for the open-source Node.js projects. It is the world's largest software library registry, and it is also known as a node package installer. NPM consists of two main factors: first, a CLI tool for publishing and downloading packages, and second, an online repository that hosts JavaScript packages. This repository contains over 800,000 code packages. Among those tons of packages, this project uses *xml4js* and *xml2js packages* with extending and adding some functions to the project.

Xml4js is an NPM package that provides XML to JSON with JavaScript Parser using the XML Schema conversion method. This package reads XML Schema that is consistent with all XML documents and converts XML to JSON based on the input of XML Schemas. [5] Xml2js detects and parses XML Schema and transforms JavaScript Object into a consistent schema-driven structure. [6]

3. SYSTEM DESIGN

◆ IMPORTING DATA

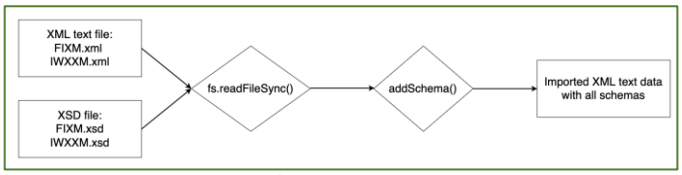

Fig 6. Importing text files and loading data to the system

Fig 6 is the flowchart of the first step of the conversion process is importing XML files to the system for loading data from imported text files. FIXM and IWXXM XML files include and import multiple schemas. In this project, input files need two different types of files: XML files and XSD files that define and verify the formats of each XML file. This system uses the fs.readFileSync() function, which is the node.js synchronous function to load the file from the filesystem and read the file's content into the system.

On the first try, I tried to write the converter code with *xml2js* library. After running the code, I could see the error message that the code could not read the included and imported multiple schemas from XSD files. Then I started to search for another library for converting XML to JSON and then *xml4js* library popped up in my research. After figuring out the *xml4js* library's functions, I combined *xml4js* and *xml2js* library to use both libraries' defined functions. I coded to use addSchema function of *xml4js* and put parseString function of *xml2js* library inside the addSchema function. Xml4js library implements finding included and imported schemas through XSD files and downloaded it and then the system reads all connected schemas and imports XML text data. After importing XML text data, the parseString() function implements to take a string using parse string and then checks whether the string

which we got is null or not. If it is null when we are changing it to an empty string for various options, and if the char is not null then it checks whether it is a function or object and then returns that based on the value to generate a string. Converting process is presented step by step in the following section in detail.

- CONVERTING DATA

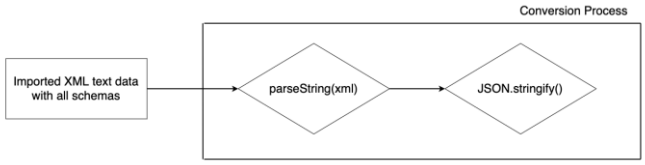

Fig 7. Converting process

Fig 7 is the second step of the conversion process is converting the imported data. As FIXM and IWXXM include multiple schemas to define, the system needs to implement the addSchema function. For Schema, there are internal references and external references. The internal references in an XSD file are present in the same XSD document and the external references use a separate XSD document. The external references contain the "*import*" and "*include*" elements. The "*import*" element allows using schema components from any schema documents with different target namespaces. Unlike "*import*", the "*include*" element allows adding all the components of an included schema from other schema documents that have the same target namespace or no specified targetnamespace to the containing schema. The big difference between "*include*" and "*import*" is from the same or different namespace.

The method to populate schemas works in two steps. First, the system finds all the files that are included and imported by the main XML Schema files. The second step is to download and add all the downloaded schema files into one with handling the namespaces. Those steps are described in the following paragraphs in detail.

First, the converter collects schema files from an XML and an XSD file that includes all schemas and their schemalocation. The Schemalocation attribute has two values, separated by a space. The namespace to use is the first value, and the second value is the location of the XML Schema to use for that namespace. In FIXM and IWXXM datasets, schemalocation is used to import and include tags and it stores all the namespaces and prefixes. First, the system finds and collects all the namespaces that are declared in the input file. And then it stores namespaces' names and prefixes. In the XML file, the system tests the "schemalocation" of attribute on the root element that is containing the location of the main schema file and put the location of the main schema into the hash. In the XSD file, the system detects the "import" and "include" elements and gets the "schemalocation" attribute of the element, and adds it to the hash. The system is designed to repeat this process when it finds the newly added schema file. Once the finding process is done, the system has a has that contains all locations of the imported schema files without any duplications.

Next, based on all collected schema files from either XML file or XSD file, the system downloads schemas through the schema URL and namespace URL and adds all the downloaded schema files into the main schema file with handling the different namespaces. In every schema file, the prefix associated with the namespace is the same as http://www.w3.org/2001/XMLSchema. Thus, the system is designed to download the prefix with this namespace and replace the element and attribute when the system finds a different namespace from "XSD". The converter finds the namespace that is given by the "targetnamespace" attribute. After finding all schemas that are included imported and verifying the defined XML formats in the XSD file, the system implements the parsing process.

Third, within a complex type of XML element, it converts the element's attributes first sequentially. Each attribute is designed to convert as a JSON property with the '$' field and each value is designed to convert with the '_' field. Using *xml4js* and *xml2js* library, the system designed five patterns for the conversion, and Table 5 is an example of conversion pattern mapping between XML and JSON.

1. An empty element
2. An element with text content
3. An element with an attribute
4. An element with text content and attribute
5. An element with multiple attributes

According to those conversion rules, the converter starts with parseString which is defined in the xml2js package. ParseString converts XML to JavaScript Object and the converter uses the JSON.stringify function to create a JSON string out of an object or value and serialized the data format into JSON parses the XML text data to export as a JSON text file.

- EXPORTING DATA

Fig 8. Exporting converted JSON file

Lastly, when the conversion process is done, there will be converted JSON text data in the system. The system implements the fs.writeFileSync function to write a file in the filesystem. Fs.writeSync function does not take a callback, which means that the execution of the script can be paused until the process is finished. Usually, node.js and developers prefer to use an asynchronous function like fs.writeFile(), but I chose to use fs.writeFileSync() because the system does not serve any new requests causing a call to this function that could have a big impact on the performance of the system.

4. SYSTEM IMPLEMENTATION

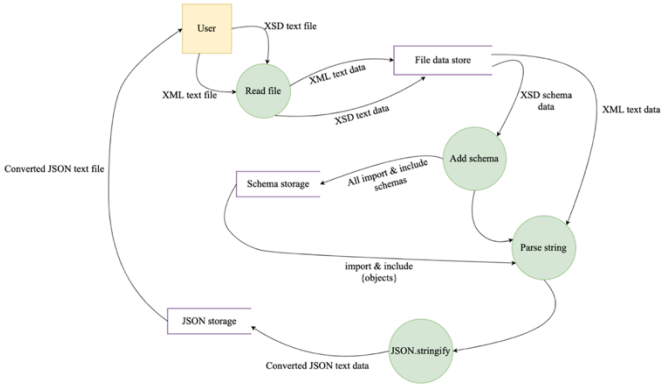

Fig 9. Data flow diagram of the converter

Fig 9 is a data flow diagram of the data converter. When the user inputs an XML file and XSD file into the system, the read file process starts importing those inputs and delivers XML text data and XSD text data to a database named File data store. First, the system performs add schema process with XSD schema data extracted from the File data store database. This process produces all import and include schemas to the Schema storage database and moves on to the parse string process. When the parsing string process starts, all found import and include schemas that are converted to object format delivered to this process. Then the process moves on to the last process called JSON.stringify and this process produces converted JSON text data to the JSON storage database. As all processes are completed, JSON storage sends a converted JSON text file to the user, then the converter system ends. Users can get converted JSON files according to the XML file format that the XSD file makes validation of it.

V. SCHEMA VALIDATION METHODOLOGY

Validator checks an XML document is well-formed and validates the following constraints are satisfied. Firstly, an XML document has its closing tag according to its opening tag, in other words for each opening tag there must be a closing tag. Second, every element must have a single parent element which means the opening and closing tags must be in the same element. Third, every XML document must have a single root element that is a single top-level element. It contains all elements and has neither a parent element nor sibling elements. Lastly, the smaller-than character '<' must not occur in an element's textual content.

1. SYSTEM ARCHITECTURE

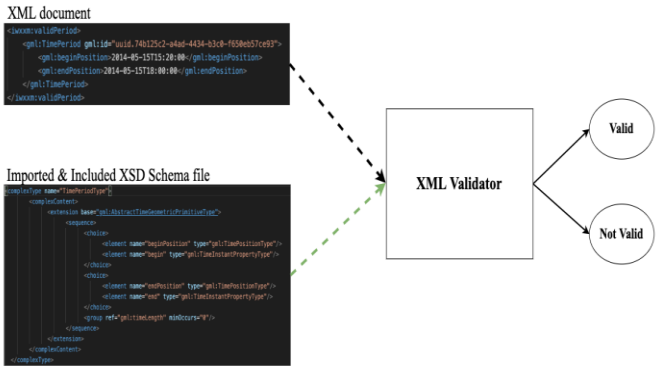

Fig 10. XML Validator Architecture

In Fig 10, this research is building an XML schema validator that validates XML files across multiple XSD schemas according to their definitions.

- Receive an XML string with the content to be evaluated.
- Scan the file for each schemaLocation.
- Compose a schema that includes all schemas.
- Validate the XML again with the new generated schema.

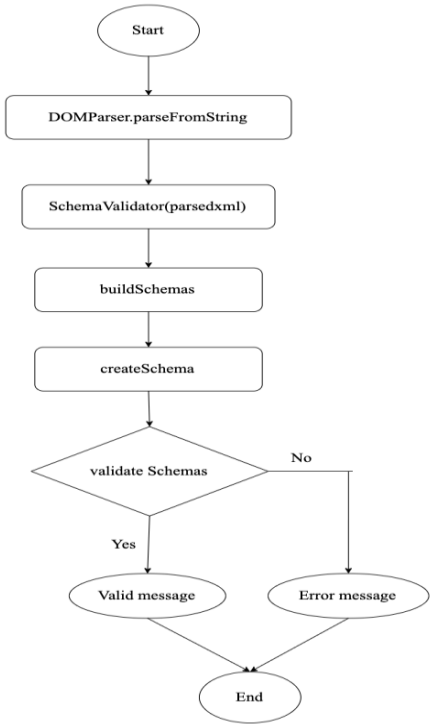

Fig 11. Flow chart of XML schema validator

In Fig 11, this is the flow chart of the XML schema validator. This starts from reading the XML file to parse using with DOMParser and creating a schema validator constructor using this XML file. Then, the system changes the schemas collection to override the schemaLocation of a specific namespace. At last, the validator validates the file with Boolean

and returns an error message or confirmed valid message. Each step is described in detail with a diagram in the following sub-sections.

2. BUILDING A TOOLKIT

- libxmljs

libxml is an NPM module for checking well-formed, error reports, validity against the XML schema, getting Xpath values, and loading XML from string or file path. [7]

- xmldom & xpath

Xpath is used for navigating through elements and attributes in an XML document. For navigating XML documents, Xpath uses path expressions and defines parts of an XML document. Downloading xmldom along with XPath, xmldom provides two interfaces: DomParser and XMLSerializer. And it also has it's SAXParser providing XMLReader and DOMHandler interfaces. [8]

- TypeScript

Typescript is an optional static programming language that is a superset of JavaScript. This language is developed for dealing with large applications and transpiles to JavaScript. The big difference between JavaScript and Typescript is whether a "type" is defined in advance or not. Type in Typescript is defined number, Boolean, string, void, null and undefined. By providing types before compiling, the compiler can find types on its own. Defining types can find errors early and make it easy to manage code. TypeScript supports Object-Oriented Programming (OOP) using classes and interfaces. By defining schema, schemas, and schema validator files, the XML Schema validator is designed with prototypical inheritance with TypeScript classes [9].

3. SYSTEM DESIGN

- RECEIVING XML STRING

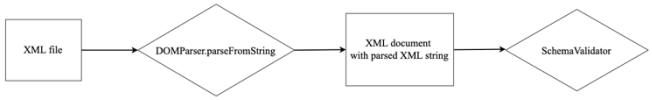

Fig 12. Receiving XML string process

DOM is the document that represents nodes and objects. Fig 19 is the process of receiving an XML file and parsing the XML string. First, the DOM parser receives an XML file and the parseFromString function of the DOM parser parses XML string into an XML document. It returns the XML document with parsed XML string and this is sent to constructor named schemaValidator. This constructor creates a schema validator using imported XML documents.

- SCANNING SCHEMALOCATION

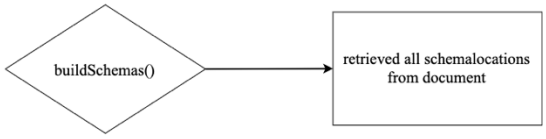

Fig 13. Scanning all schemalocation process

Xmlns is an identifier in the document that does not have to be a URI to the schema and xsi:schemaLocation is giving information to the actual schema location. BuildSchemas() function checks whether the namespace is registered or not and gets all the xsi:schemaLocation attributes in the imported document. So, it retrieves a list of namespaces based on the schemaLocation attribute and creates a schema's collection from the content of a schema location.

- COMPOSING ALL SCHEMAS

Fig 14. Generating the new schema process

With all retrieved schemas, schemas.create() function changes schema collection to override the schema location of a specific namespace. After the process, this creates a new schema and inserts it into the collection by inputting schema location and namespace in the function attributes.

- VALIDATING XML WITH SCHEMAS

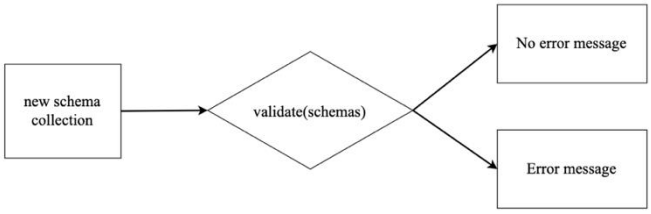

Fig 15. Validating XML with the new schema collection

Lastly, the validator validates the XML file again with the newly generated schema collection. It validates against a list of schemas by returning the XML of XSD that includes all the namespaces with the local location to build the unique import schema. The validate function checks the schemas with an XML file and returns with Boolean. If it returns true, it prints valid file message, otherwise, it prints an error message to the console.

- EXCEPTION HANDLING

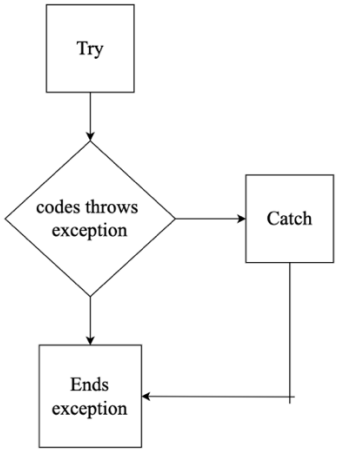

Fig16. Error handling of XML schema validator

There is a difference between error and exception. Error is a major cause that occurs when the program is running and is abnormally ended. Error is severe to the system that cannot be recovered when there are memory shortage or stack overflow. On the other hand, the exception is runtime errors that happen during the program is running and can be handled easier than errors. As in Fig 16, the XML schema validator has exception handling with a try-catch statement. Validating against a list of the created schema is the code for the try statement that will run for handling exceptions, and the catch statement defines the error message that will print on the console when the exception happens.

4. SYSTEM ARCHITECTURE

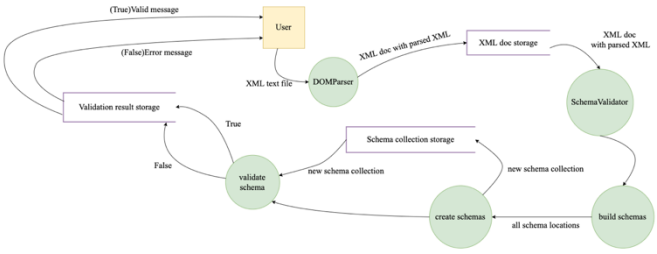

Fig 17. Data flow diagram of schema validator

In Fig 17, when the user inputs an XML file into DOM Parser, this process starts the parseFromString function to make a DOM document with parsed XML and stored it in DOM doc temporary storage. Then, a schema validator is made with this DOM document, and the process moves to build schemas with schema collection. In the building schemas process, it collects all schema locations in the file and sends them to the next process, which is creating schemas. Based on collecting all schema locations, this process makes the new schema collection and inserts it into the storage called schema collection storage. Next, the validator starts validating the schema according to the new schema collection and returns true and false for the validation output. These Boolean results move on to the storage named validation result storage and it outputs the result as a message to the user. Users can get either a valid message or an error message after all validation processes are completed.

VI. BUSINESS RULE VALIDATION METHODOLOGY

Business rules are the rules that define or constrain some aspects of business and are set to either true or false. This can apply to people, processes, corporate behaviors, or computing systems and it is intended to influence the behavior of the business to help the organization achieve its goals. Business rules are formalized in a language that managers and technologists can understand. Business rules exist for an organization whether or not they are ever written down, talked about, or even part of the organization's consciousness. Building a business rule develops a strategy to provide high-level direction about what an organization should do and business rules to provide detailed guidance about how a strategy can be translated into action.

The criteria for FFICE Business Rule Validation are provided by the FIXM committee. In simple terms, several FFICM XML Elements need to exist in the message for it to be business rule valid. The idea of Business Rule Validation of the flight plan is when XML messages come in and before it goes out, first, the message is checked against the schema to make sure it is valid, which means that checking the data is in the right place, data values are there and matching the right pattern. Second, the message is checked against the business rules, which means to make sure that all the required elements do exist within the message.

FFICE (Flight and Flow Information for a Collaborative Environment) defines all requirements for flight plan information. FFICE data supports the ATM industry to achieve strategic and tactical performance management by emphasizing the need for data sharing to improve efficiency and benefits to the ATM industry. The data exchange of flight and flow information is a potentially promising system for the present and future ATM industry. The ATM focuses on the interoperable, integration, and harmony of a system for all ATM stakeholders and users in flight. ATM aims to increase efficiency and user ease of use and improve safety in the future ATM system. FFICE is established to facilitate improvements and solve limitations that the current system has. FFICE aims to increase collaborative communication of flight planning among ATM actors to provide facilities for real-time data exchange. Also, it aims to increase flexibility for optimal ATM management. For accomplishing their aims, FFICE released the requirements to adjust or remove the limitations that the present flight plan information has and to accommodate the goals of the future flight plan that is detailed in the ICAO FF-ICE document [10].

In FFICE files, various elements represent flight planning. Each element has different criteria, different classification alphabets, and regulations. To validate that each element has the right information and meets the criteria of

information, checking the contents of the flight plan is the priority before building an XML validator.

1. FLIGHT PLANNING ELEMENTS

- Aircraft Address

Aircraft address is *six* alphanumeric characters. It provides a digital aircraft identification number and is used to link information for flight notification for tracking aircraft position information received via ADS-B for air traffic system [11].

- Aircraft Type Designator

An aircraft type designator is a *two, three, or four* alphanumeric character. It designates every aircraft type and subtype that presents in flight planning. The aircraft type designator is used by air traffic control and airline operations in flight planning. Providing accurate aircraft type designator information can make efficient updating information through the website [12].

- Wake Turbulence Category

ICAO wake turbulence category is divided into four categories: *J, H, M, and L* based on the maximum certified take-off mass [13].

- Registration

Aircraft registration defines a code unique to a single aircraft. It indicates the aircraft's information including aircraft's country of registration and an aircraft can only have one registration. In the United States, the registration number starts from 'N'. N-number begins from 1 to 9 and I or O cannot be used with because the similarities with numerals 1 and 0. This project is designed to check and validate the registration number between *N100~N999, N1000~N999, N10000~N99999, N1A~N9Z, N10A~N99Z, N100A~N999Z, N1000A~N9999Z, N1AA~N9ZZ, N10AA~N99ZZ, and N100AA~N999ZZ* [14].

- Location Indicator

ICAO airport codes or location indicators are *four-letter* codes that identify aerodromes around the globe. It is used for flight planning by air traffic control and airline operations and for identifying aviation facilities and flight information regions. This code is different from IATA airport code [15].

- Comm/Datalink/Nav Capability Code

Those code describe the communication (COM), navigation (NAV) and datalink of aircraft. It is a list of letters and numbers that describes comm and navigation equipment. Those number and letter combination code are defined in ICAO codes, so this validator checks whether the code is valid or not based on this list. A, B, C, D, E1, E2, E3, F, G, H, I, J1, J2, J3, J4, J5, J6, J7, K, L, M1, M2, M3, N, O, P1-P9, Q, R, S, T, U, V, W, X, Y, Z are used for communication, datalink, and navigation capability code in flight planning [16].

- Flight Type

The type of flight indicates what kind of type does each flight has. It has five different purpose of flight type. *'G'* means general aviation, *'S'* means scheduled air service, *'N'* means non-scheduled air transport operation, *'M'* means military and *'X'* mean other than any of the defined categories above.

- Gufi

Gufi stands for Globally Unique Flight Identifier sends any flight data to any ATM system before a flight. This consists of 36 characters in total and gufi code is divided in five parts with a designated how many characters are required. Those characters are comprised of eight – four – four – four – twelve characters in a row. This should be checked through business rule validator.

2. BUSINESS RULES OF FLIGHT PLAN

Before developing the validation, the document types, page types, and fields need to be reviewed to identify the fields to capture and to determine the other business requirements. After I examined each sample page, the next step was identifying the fields of interest. Each variant of a page type includes all of these fields, but the position of each field is different for each variant. Table 6 is the list summarizing the fields that I need to capture for each page type of each document type. This research deals with 10 different FFICE XML files including AgreedTrajectory, EDeparture, EnrouteStatus, File, FileUpdate, FilingStatus, RevisionRequest, SubResp, TrialRequest and TrialResponse. Each file has different page types and fields.

The first step was defining the structure of each document type and the fields that I want to capture from each page. The next step is setting the business rules by defining what elements need to exist in the message for a valid message and how I want to validate the captured data to determine whether the data meets the business requirements. According to the comparison between FIXM v.4.1 and v.4.2, the FIXM committee presented added files and new items in FIXM 4.2. The below criteria define the new version of FIXM and those are the business rules for validation.

- FIXM v.4.2 root element contains application header and Schema location
- Identifier and identifier domain is added in the originator, gufiOriginator, and recipient element.
- Under flight element, gufiOriginator element with identifier and identifier domain is added.
- RelevantAtmServiceProvider element is added with provider and providerType element. This element only exists in AgreedTrajectory, File, FileUpdate, RevisionRequest and TrialRequest XML files.
- In the provider element except for the FileUpdate file, sub-elements should be defined as necessary to identify the provider. Sub elements consist of contact with name and title sub-elements, identifier, identifierDomain, and name element.
- In the provider element in the FileUpdate file, sub-elements consist of a name and onlineContact with email, linkage, and network element, which has a type sub-element, and title, identifier, identifier, and name element.
- Ffice message type is added
- Type element is added that is proposed using the same message type in the header and FIXM 4.2 XML files.

VII. Conclusion & Future Work

Keeping the air traffic system updated is a very important issue in the aviation industry for efficient data sharing. This research project investigates developing the converter and validator including XML schema validator and business rule validator of flight planning with JavaScript and TypeScript. This research project presents an experimental approach to using the new programming language in developing data converters and validators for the ATM industry. The report detailed the toolkits, system architecture, and system design with each process of convertor and validator. My approach is based on utilizing Node.js with JavaScript and TypeScript to achieve ease of use and development.

Furthermore, it should be interesting to perform the comparative analysis of schema validator and business rule validation between systems with Java and TypeScript about efficiency and maintenance. To compare the efficiency of converter between those languages, first, I must build another converter in Java with the current version data files of FIXM and IWXXM. Next, then I can conduct the comparison between Java and Node.js system with for the efficiency. Also, this research will be conducted more efforts on completing business rule validation for the future work.

Acknowledgment

Throughout my entire Master's study, I have been supported, inspired, and motivated by many people. I would like to particularly acknowledge those who have directly supported my Master's research work.

First, I would like to sincerely thank my advisor Dr. Massood Towhidnejad and Dr. Timothy Wilson who brought me to the Embry Riddle Aeronautical University when I did feel lost after a failure of my first Master in France. Since then, they have been guiding me throughout my research and inspiring my life. Neither this thesis work nor any of my future research work in the software engineering field would be possible without their guidance.

I would like to acknowledge my Nextgen ERAU Applied Research (NEAR) lab members Mr. Ramin Rashedi and Mr. Carlos Castro for both giving me an awesome chance to work at this wonderful research lab and for insightful comments on my research work.

I am grateful to all my faculty members, staff, and colleagues at the Software Engineering Department. I especially Tim Elvira, Juan Ortiz Couder, Michael Fornito, Ross Dickinson, and Lauren Meyer for encouraging and supporting me throughout my Master course. They have set models of how to be a good student for me. Spending my master's time together with them is an unforgettable memory in my life.

I would like to thank all my family members especially my father for their support, patience, trust and love. And my friends especially Soonie Sooyeon Jo, Seungwon Oh, Yoonie Yoonjung Joo, Myeongwon Song, Jihye Kwon, Jisoo Yoon, Shinyoung Joo, Danny Dongmin Na, Min and Duhyun Kong. Even though I am eleven thousands of miles apart from my home, I never feel lonely because of them.

My completion of this project could not have been accomplished without the support of all. I say a big thank you to you and love you all ☺♡

A List of Acronyms

Abbreviation	Acronyms
AIXM	Aeronautical Information eXchange Model
ATFM	Air Traffic Flow Management
ATM	Air Traffic Management
CHR	Constraint Handling Rules
DOM	Document Object Model
DTD	Document Type Definition
FF-ICE	Flight and Flow Information for a Collaborative Environment
FIXM	Flight Information eXchange Model
FLXM	Flow Information eXchange Model
HTML	HyperText Markup Language
ICAO	International Civil Aviation Organization
IWXXM	ICAO Meteorological Information eXchange Model
JAXB	Java Architecture for XML Binding
METAR	Meterological Terminal Aviation Routine Weather Report
NEAR	Nextgen ERAU Applied Research
NOTAM	Notices to Airmen
OOP	Object Oriented Programming
SAX	Simple API for XML parsing
SIGMET	Significant METeorological Information
SOA	Service Oriented Architecture
SSL	Secure Socket Layer
SWIM	System Wide Information Management
TAF	Terminal Aerodrome Forecast

W3C	World Wide Web Consortium
XPath	XML Path Language
XSD	XML Schema Definition
XSLT	eXtensible Stylesheet Language Transformation
XML	eXtensible Markup Language

REFERENCES

[1] FAA NextGen website, Available: https://www.faa.gov/nextgen/

[2] SESAR and FAA, B.Taylor(Lincoln Lab), M.Tanino (FAA) and H.Lepori (Eurocontrol/SESAR)"FIXM Developer's Manual", Nov 2012, Edition: 1.1, Available: https://www.fixm.aero/releases/FIXM-1.1/FIXM_Core_v1_1_Modeller_Manual.pdf

[3] FIXM website, Available: https://fixm.aero/

[4] Node.js website, Available: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

[5] Xml4js NPM module library, Available: https://www.npmjs.com/package/xml4js

[6] Xml2js NPM module library, Available: https://www.npmjs.com/package/xml2js

[7] Libxmljs2 module library, Available: https://www.npmjs.com/package/libxmljs2

[8] @xmldom/xmldom module library, Available: https://www.npmjs.com/package/@xmldom/xmldom

[9] Adam Freeman, "Essential Typescript4: From Beginner to Pro" (Book), Apr 2021, Apress, Available: https://learning.oreilly.com/library/view/essential-typescript-4/9781484270110/

[10] Manual on Flight and Flow – Information for a Collaborative Environment (FF-ICE), Available: https://www.icao.int/airnavigation/IMP/Documents/Doc%209965%20-%20Manual%20on%20FF-ICE.pdf

[11] ICAO Aircraft address, Available: https://www.iomaircraftregistry.com/flight-operations/flight-operations/icao-24-bit-aircraft-address-mode-s-coding/

[12] Aircraft Type Designators: https://en.wikipedia.org/wiki/List_of_aircraft_type_designators

[13] Wake Turbulence Category: https://skybrary.aero/articles/icao-wake-turbulence-category

[14] Registration: https://en.wikipedia.org/wiki/Aircraft_registration

[15] Airport code: https://en.wikipedia.org/wiki/Lists_of_airports_by_IATA_and_ICAO_code

[16] Communication/datalink/surveillance capability code: https://en.wikipedia.org/wiki/Equipment_codes

[17] XML Schem Validator with PHP: https://github.com/eclipxe13/XmlSchemaValidator

All resources and code are available: **https://gitlab.com/nearlab/swim/grp**

Sherry Jungwon Zang's Github: https://github.com/jungwonsherryzang?tab=repositories