GenA Senior Design

Final Report

Team 45

Collins Aerospace - David Lempia

Dr. Miner

Kyle Smoot

Mason Ogborne

Chris Olsen

Mohan Zhong

Morgan Smith

sdmay21-45@iastate.edu

Revised:4/25/2021

Executive Summary

Development Standards & Practices Used

- Ada 95 Standard
- Google Java Style Guide
- XML 1.0 (W3C)
- Collins Ada Standard

Summary of Requirements

We will develop a code generation tool for Collins Aerospace. This code will generate Ada code. The generated code should be functional and provide error messages. We will use a high-level language to develop the tool.

Applicable courses from Iowa State University Curriculum

- COMS 227
- COMS 228
- COMS 309
- COMS 319

New skills/knowledge acquired that was not taught in courses

- Code generation
- Ada code
- XML code

Table of Contents

Figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

We would like to thank Collins Aerospace for providing this project and supporting sample code/documentation. We would like to extend a special thanks to David Lempia, Stan Thayer, and Kathleen Knot for taking time to meet with and assist us regularly. We could not have done it without them.

1.2 Problem and project statement

Collins Aerospace uses Network Data Objects (NDOs) for avionics systems to communicate with each other. An NDO is defined using a strict XML schema, specifying IP addresses, data types, memory addresses, and other critical information about the message. Once this schema is created, a developer uses the information in the schema to implement the message in Ada. This is a tedious process, and is prone to errors. When these errors occur, they require significant time and effort to debug and correct. Collins Aerospace is looking for a solution to produce the NDOs in a manner that is faster and less prone to errors.

Our solution approach is to create a code generator to create the NDO messages directly from the XML schema. It will use the XML file produced by Collins as part of their requirements process. It will create the ADA file directly. This will reduce the time needed to create the file, as well as reduce small errors in the code. This project consists of producing the generator itself, as well as the documentation for the project.

1.3 Operational Environment

This project is to run inside the command line. Thus, the environment is anything that can run Java. Given the wide array of devices that satisfy this criteria, no special consideration is needed.

1.4 Requirements

1. GenA shall provide errors if XML data needed to generate software is missing from the xml files.
2. GenA shall generate Ada Software in the Ada 95 standard (ISO-8652:1995)
3. GenA shall generate software to read and write NDO messages

a. The generated Ada software shall provide a procedure to read in NDO messages.
b. The generated Ada software shall provide a procedure to transmit NDO messages.
c. The generated Ada software shall provide a method to access the NDO messages.

4. Collins Engineers shall be able to run GenA and associated options in Windows 10 from the command prompt.
   a. GenA options shall include:
      - The name of the XML file(s)
      - The name of the tailoring mechanism (e.g. a template file name).
5. Collins Engineers shall be able to tailor the way Ada software is generated. *Note: This is desired for growth and maintenance of the GenA tool.*
   a. Tailoring of GenA shall be able to generate software in different languages such as c or c++. *Note: this is desired of use of GenA in different products.*
   b. Tailoring of GenA shall be able to define new types of I/O.

1.5 Intended Users and Uses

- Collins Engineers - Code Generation
  - Engineers using the generator to create NDOs
- Collins Engineers - Maintainers
  - Engineers who will update the generator in the future, as requirements and specifications evolve

1.6 Assumptions and Limitations

1.6a Assumptions

- The generator will be run from the Windows 10 Command Line
- The generator will only supports one user per running instance
- The generator is only to be used by Collins Aerospace

1.6b Limitations

- The generated code cannot be fully tested by the senior design team
- Due to security constraints, the team cannot access all related materials
- Time constraints due to the project having a hard end date at the end of the spring semester

1.7 Expected End Product and Deliverables

1. GenA Ada Code Generator

This is the executable generator created for Collins Aerospace. It will be delivered in a form that allows the generator to be run immediately upon reception of the generator. It will be used for generating Ada code as part of the NDO creation workflow.

2. Generator Use Documentation
   Documentation will be provided to Collins concerning the use of the generator. It covers command line options, available generators, output format, and other generation options. It also covers installation and running instructions.
3. Source code and Design documentation
   This is the source code for the generator. It will also include documentation on the design for the generator, design choices, and information needed to maintain the generator. Documentation will also be provided for extending the generator, such as adding another generator or message type.

2 Design

2.1 Previous Work and Literature

An Engineer at Collins has developed a simple code generator. It is limited to code generation in Ada, and does not work for all use cases. It does not generate all files required. It is also temperamental when used. There are code generation tools and XML/Ada code examples.

Other open-source generators were surveyed for applicability to the project. The OpenAPI generator, Yeoman, hygen, and JHipster were surveyed. Amongst these, the OpenAPI generator most closely matched our use case. We used this as a pattern to base our generator off of.

2.2 Design Changes

The design has changed somewhat since the final design document from SE 491. The validation module has been completely removed from the design. This is due to Collins already having a tool to validate their XML files. Their tool is more complete than any implementation we could create due to confidential knowledge that Collins has about the XML schema. Due to this, it was determined that the validation module would provide no significantly new functionality, and we thus removed.

2.3 Design

The design consists of seven primary modules. The first is the core module. The core module is responsible for orchestrating operations between modules. The next module is the configuration module. This module is responsible for adding language and message specific settings for the generation. The third module, the generation module, is responsible for getting the raw data from the parsing module, putting the data in the proper format, calculating derived values and then passing it to the templating module. The parsing module reads the XML file, and puts the data into java objects. The templating module takes the processed data and produces the contents of a file using a templating engine. It is also responsible for writing the file to disk. For this project, we plan on using Handlebars for templating . The common module is a simple module containing resources needed by multiple modules, such as the Java models needed for parsing and generation. The final module is the CLI. It is responsible for sending arguments to the configuration module. Having a separate CLI modular allows for the possibility of other interfaces in the future with minimum refactoring.

A standard generation workflow would look something like this. A user would start a generation with the CLI. The configuration module would provide a configuration based upon the program arguments. This would be passed to the core module. The core module then calls the parsing module to parse the file. The parsing module returns java objects counting the information in the XML file. The core module would send this data to the generation module. The generation module would then process the data, then pass it to the templating module. The templating module would then create the file contents using Handlebars, then write it to file.

This was designed largely with maintainability and extensibility in mind. There are many ways to produce a file, but a major requirement of this project is to be extensible as needs change in the future. The client has stated that the code being generated is guaranteed to change over time. The breakdown of modules allows the project to be easily extended in the future.

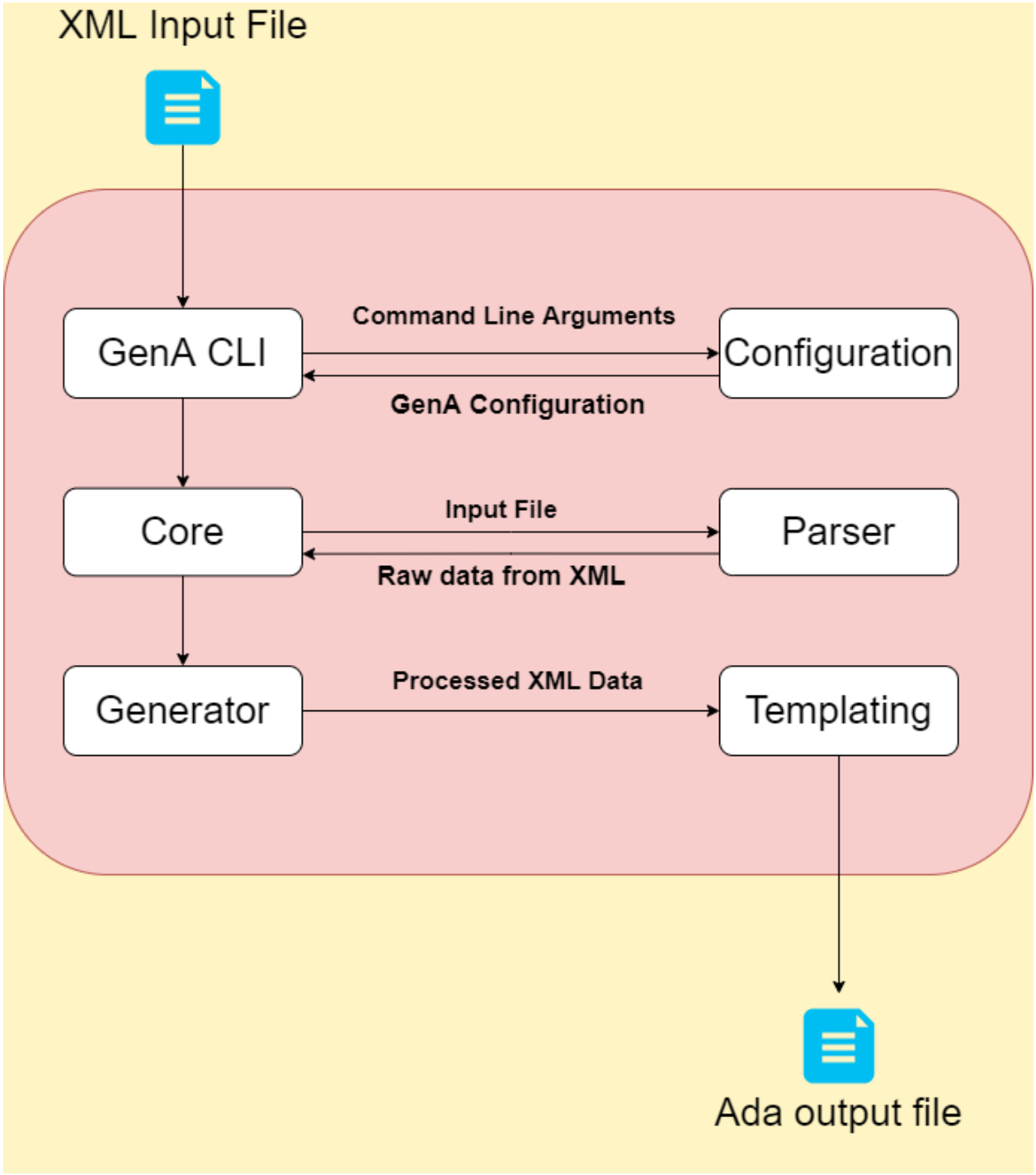


Figure 2. Module Diagram

2.4 Technology Considerations

We will be using java as the primary language. This allows the project to be easily maintained, due the prolific knowledge of java. One drawback of this is that the project, while written in java, will

also require a knowledge of Ada. The Java libraries Apache Commons-CLI and Handlebars were used.

2.5 Security

Security is not a consideration for this project from a software perspective. This project will only be used in-house by collins aerospace engineers.

3 Testing

3.1 Unit Testing

Code is tested using Junit to ensure requirements compliance. Modules contain their own tests, and do not test the functionality of other modules. When functions from other modules are required, Mockito is used to mock the functionality of these modules.

3.2 User Acceptance Testing

Acceptance testing provides some challenges. Due to security concerns, the team will not be able to test our generated code in its final environment. As a result, to test the functionality of the project, the client took delivery of the GenA project at regular intervals to test and evaluate. This allowed the project to be tested more stringently on real-world code.

3.3 Integration Testing

Given the above listed issues with testing real world code, the client sent mock examples to be used for testing. In addition to code correctness, the client determined that a satisfactory test for the project would be the generated code compiling with no modification to the generated code. Therefore, for this project, the generated code compiling suffices for an Integration test.

3.4 Results

For the GenA codebase, all tests pass. Additionally, all modules interface properly with each other. From the integration perspective, the generated compiles under gcc, which was the primary condition for success given by the client. User acceptance testing has gone well, with the client

being able to use the tool as intended. Small changes have been requested, but these center around small quality of life improvements and minor templating changes, rather than large scale issues with program functionality. Overall, GenA conforms to it's requirements.

4 Implementation

GenA was implemented in Java. It utilizes Gradle as a build tool. Gradle was selected for it's ease of use when setting up multi-module builds. A gradle task produces a FatJar, which contains all required dependencies to run GenA from the command line. This allows GenA to be run on any device that supports Java, ensuring high portability for the project.

GenA uses Handlebars for it's templating engine. Handlebars was chosen for its simple syntax, but with more powerful functions than its predecessor, Mustache.

The project currently operates as a Command Line Interface.

An example of Ada generated from XML input is attached in Appendix IV.

5 Closing Material

5.1 Conclusion

In this project, many plans and baseline development has been completed to provide the best possible base for the development of our project throughout the rest of this semester and in the next semester. We have laid a sustainable task management system, group discussion processes and schedules, and have deeply discussed the requirements and goals of this project.

GenA, the proposed project in conjunction with Collins Aerospace, proposes using NDOs for use in avionics systems as a means of intercommunication between systems. This must be completed through a strict schema already used within this system to send messages between system subsets. This schema is then used to implement the message in Ada. However, because of the drawbacks of

this current system such as errors, time consumption, and correction effort, Collins Aerospace wishes to produce NDOs more efficiently and effectively.

The solution to this problem proposed by our team will implement a code generator to create NDO messages from the XML schema provided using the XML file already being produced during the Collins Aerospace currently in use processes. This will then create the ADA file directly, reducing the potential for errors, and providing a more efficient, and understandable process to work with. This solution will greatly improve on the current system, and produce a much more user friendly, and efficient experience.

5.2 References

N/A

5.3 Appendices

5.3.1 Appendix I: Operators Manual

Basic GenA Operation:

Build:

GenA uses gradle as a build tool. If gradle is not installed, the project includes a gradle wrapper, gredlew. Runing gradle build or gradlew build will build the project.

Jar:

To produce a Jar File with all needed dependencies, run gradle shadowjar from the root project directory. The created jar will be in the path modules/gena-cli/build/libs. The jar file is gena-cli-all.jar.

CLI:

usage: java -jar gen-cli-all.jar [-h] -i <arg> -o <arg>

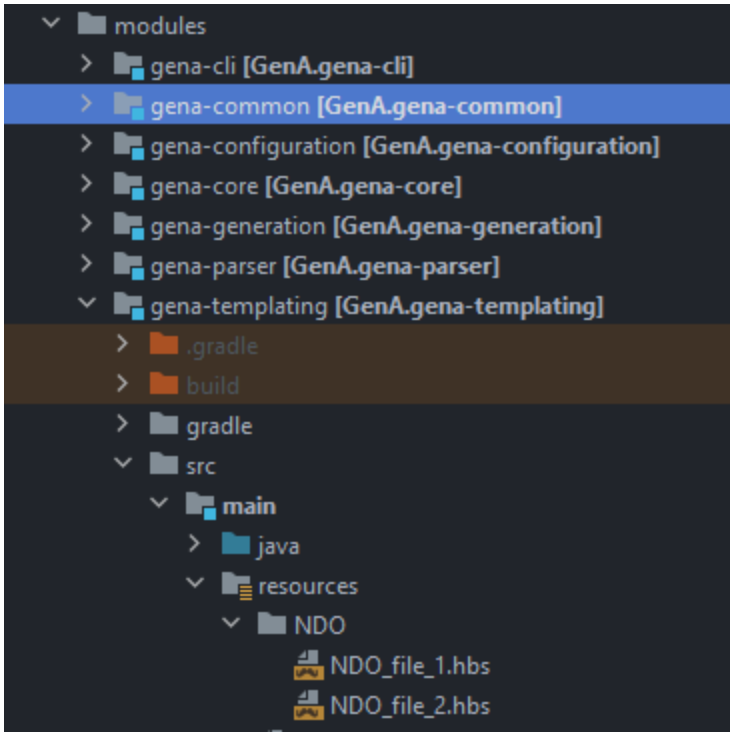
-h,--help help

-i,--input <arg> input filename -- Required

-o,--output <arg> output filename -- Required

Modifying Templates:

If you need to change the templates for the code generation, they are under the resources directory in the gena-templating module.



Edit the template that needs to be changed, then run gradle clean build to rebuild the project with the modified jar.

Adding templates:

To add a template navigate to the directory where the templates are stored. Add the template. The template method in the template class in the templating module will need to be modified to include the new template.

Handlebars Resources:

Official Handlebars Manual: NOTE: Handlebars is originally for JavaScript. GenA uses an equivalent java implementation.

Common Handlebars Expressions:

#each Iterates over a list of objects and applies templating to all items in list

#unless logical not

@last Helper function that returns if item is last element in list. Useful for defining comma separated values with a different character required at the end, such as a parentheses.

5.3.2 Appendix II: Previous Design Document

GenA Senior Design

Design Document

Team 45

Collins Aerospace - David Lempia

Dr. Tyagi

Kyle Smoot

Mason Ogborne

Chris Olsen

Mohan Zhong

Morgan Smith

sdmay21-45@iastate.edu

Revised: 11/15/2020 V3

Executive Summary

Development Standards & Practices Used

We will follow IEEE standards for software development. Our project will include documentation both in code and on a project level. We will comply with Collins' ADA standards. We will provide version control and comprehensive testing.

Summary of Requirements

We will develop a code generation tool for Collins Aerospace. This code will generate Ada code. The generated code should be functional and provide error messages. We will use a high-level language to develop the tool.

Applicable courses from Iowa State University Curriculum

- COMS 227
- COMS 228
- COMS 309
- COMS 319

New skills/knowledge acquired that was not taught in courses

- Code generation
- Ada code
- XML code

Table of Contents

Figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

We would like to thank Collins Aerospace for providing this project and supporting sample code/documentation. We would like to extend a special thanks to David Lempia, Stan Thayer, and Kathleen Knot for taking time to meet with and assist us regularly. We could not have done it without them.

1.2 Problem and project statement

Collins Aerospace uses Network Data Objects (NDOs) for avionics systems to communicate with each other. An NDO is defined using a strict XML schema, specifying IP addresses, data types, memory addresses, and other critical information about the message. Once this schema is created, a developer uses the information in the schema to implement the message in Ada. This is a tedious process, and is prone to errors. When these errors occur, they require significant time and effort to debug and correct. Collins Aerospace is looking for a solution to produce the NDOs in a manner that is faster and less prone to errors.

Our solution approach is to create a code generator to create the NDO messages directly from the XML schema. It will use the XML file produced by Collins as part of their requirements process. It will create the ADA file directly. This will reduce the time needed to create the file, as well as reduce small errors in the code. This project consists of producing the generator itself, as well as the documentation for the project.

1.3 Operational Environment

This project is to run inside the command line. Thus, the environment is anything that can run Java. Given the wide array of devices that satisfy this criteria, no special consideration is needed.

1.4 Requirements

1. GenA shall provide errors if XML data needed to generate software is missing from the xml files.
2. GenA shall generate Ada Software in the Ada 95 standard (ISO-8652:1995)
3. GenA shall generate software to read and write NDO messages

a. The generated Ada software shall provide a procedure to read in NDO messages.
b. The generated Ada software shall provide a procedure to transmit NDO messages.
c. The generated Ada software shall provide a method to access the NDO messages.

4. Collins Engineers shall be able to run GenA and associated options in Windows 10 from the command prompt.
   a. GenA options shall include:
      - The name of the XML file(s)
      - The name of the tailoring mechanism (e.g. a template file name).
5. Collins Engineers shall be able to tailor the way Ada software is generated. *Note: This is desired for growth and maintenance of the GenA tool.*
   a. Tailoring of GenA shall be able to generate software in different languages such as c or c++. *Note: this is desired of use of GenA in different products.*
   b. Tailoring of GenA shall be able to define new types of I/O.

1.5 Intended Users and Uses

- Collins Engineers - Code Generation
  - Engineers using the generator to create NDOs
- Collins Engineers - Maintainers
  - Engineers who will update the generator in the future, as requirements and specifications evolve

1.6 Assumptions and Limitations

1.6a Assumptions

- The generator will be run from the Windows 10 Command Line
- The generator will only supports one user per running instance
- The generator is only to be used by Collins Aerospace

1.6b Limitations

- The generated code cannot be fully tested by the senior design team
- Due to security constraints, the team cannot access all related materials
- Time constraints due to the project having a hard end date at the end of the spring semester

1.7 Expected End Product and Deliverables

1. GenA Ada Code Generator

This is the executable generator created for Collins Aerospace. It will be delivered in a form that allows the generator to be run immediately upon reception of the generator. It will be used for generating Ada code as part of the NDO creation workflow.

2. Generator Use Documentation

   Documentation will be provided to Collins concerning the use of the generator. It covers command line options, available generators, output format, and other generation options. It also covers installation and running instructions.

3. Source code and Design documentation

   This is the source code for the generator. It will also include documentation on the design for the generator, design choices, and information needed to maintain the generator. Documentation will also be provided for extending the generator, such as adding another generator or message type.

2 Project Plan

2.1 Task Decomposition

Below is an outline of the tentative technical tasks and scheduling.

1. Requirements Elicitation
    a. Regular meetings with Collins Aerospace research team
    b. Prioritize deliverables based on timeline estimates
    c. Finalize project plan and schedule
    d. Research ADA language syntax
    e. Research NDO message syntax
2. Project Plan Initiation
    a. Train developers for their specific roles
    b. Agile Development
        i. Sprint planning
3. Implement Software XML Parsing
    a. Integrate Windows 10 command prompt
        i. Create shell program
        ii. Implement XML file(s) loading
    b. Implement Parsing of XML Schema.
        i. Trim and reformat XML file
        ii. Parse data categorically
        iii. Save data to be translated
4. Implement ADA Generation
    a. Interpret and translate data

b. NDO Message Implementation
    i. NDO message I/O
        1. Module to read in NDO
        2. Transmit NDO module
        3. Access NDO module
    ii. GenA documentation
c. Error Correction Devices
    i. Device 1: Fills missing non-crucial XML data
    ii. Device 2: Outputs Descriptive Error Messaging
        1. Missing crucial data
        2. Corrupt XML schema
        3. Formatted NDO message errors
d. Tailoring Options
    i. User Defined Input Options
        1. Name of the XML file(s)
        2. Tailoring mechanism template
    ii. User Defined Output Options
        1. Output language modulation

5. System Level Testing
    a. Functionality for the Code Generation engineers
6. Implement UI
7. Additional Implementations

2.2 Risks and Risk Management/Mitigation

Risk	Gold Plating	Developer Parody	Incompatible Framework	System Testing Failure
Risk Description	Over the course of development, along with being a scheduling risk, this risk involves the scope creep of the project to surpass the allotted time, and the total cost, of development.	During development, if the developers do not stay at the same pace when programming deliverables. This is problematic because it is imperative to maintain critical due dates.	The framework chosen to program the project ends up not being compatible with all of the deliverables. This would require another language to be integrated or replaced mid-development.	The project would fail the final system testing with the customer if this risk were to occur. Failure would be due to a significant lack of features missing, not working, or breaking the program.

Risk Category (Cost, Scheduling, Technical)	Cost	Scheduling	Technical	Technical
Occurrence Indicator	The cause of this risk is the continued feature creep of developing new features, and would occur when said features take away from the other established portions of the project.	Regular qualitative review can determine the schedule of the programming pace compared to the planned deliverable dates.	Quantitative testing and proper planning should both prevent and expose this risk. Ideally it would be identifiable early in development with it becoming more dangerous late into development.	3-4 weeks prior to the scheduled delivery. If quality testing occurs on both the user and customer side, this risk will become evident.
Impact Rating (Negligible - Catastrophic)	Minor	Minor	Serious	Critical
Occurrence Rating (Improbable - Frequent)	Occasional	Probable	Improbable	Remote
Risk Exposure Rating (Low - High)	Medium	High	Low	High
Response Type (Avoid, Mitigate, Accept, Transfer)	Accept	Mitigate	Transfer	Avoid
Response Strategy	The response strategy to this particular risk is through having a predefined development scope and schedule so that if any feature creep occurs, it happens when all main deliverables are complete.	The strategy to mitigate this risk would lie primarily in the development of communication skills, project roles, and the relationship between developers and the client.	If the project required that, if the current framework proved to be insufficient, then a change of language or framework would be a worthy response type to transfer the risk.	Regular review and analysis of the current development process with regular testing of the current product. This is supplemented with a rigid scope and good version control practices.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Many potential milestones and metrics have been discussed within the group as well as with the project sponsor. Weekly progress updates and meetings will help to evaluate the progress of each individual as well as the project as a whole. Using trello, github commits and other task management tools will allow the team to track progress of tasks and development within the project. We have created project milestones such as reading basic XML code, passing it to the generator and have set a timeline for those.

2.4 Project Timeline/Schedule

This project will follow the following predetermined schedule as part of the course requirements set by the instructors.

- Sept 29 - Team Website V1
- Oct 1 - Design Document V1
    - Chapters 1 & 2 focused
- Oct 22 - Design Document V2
    - Chapters 3 & 4 focused
- Nov 12 - Final Design
    - Chapters 5 & 6 focused
- Nov 16/23 - Demo to client
- Bi-weekly project reports

In addition to these set requirements, as a team we will create further small milestones and timeline markers in order to complete the project. Our currently proposed scheduling includes weekly meetings and progress reports as a team as well as between the team and sponsors. Furthermore, this project will continue as a second semester with COM S 492 in the spring semester.

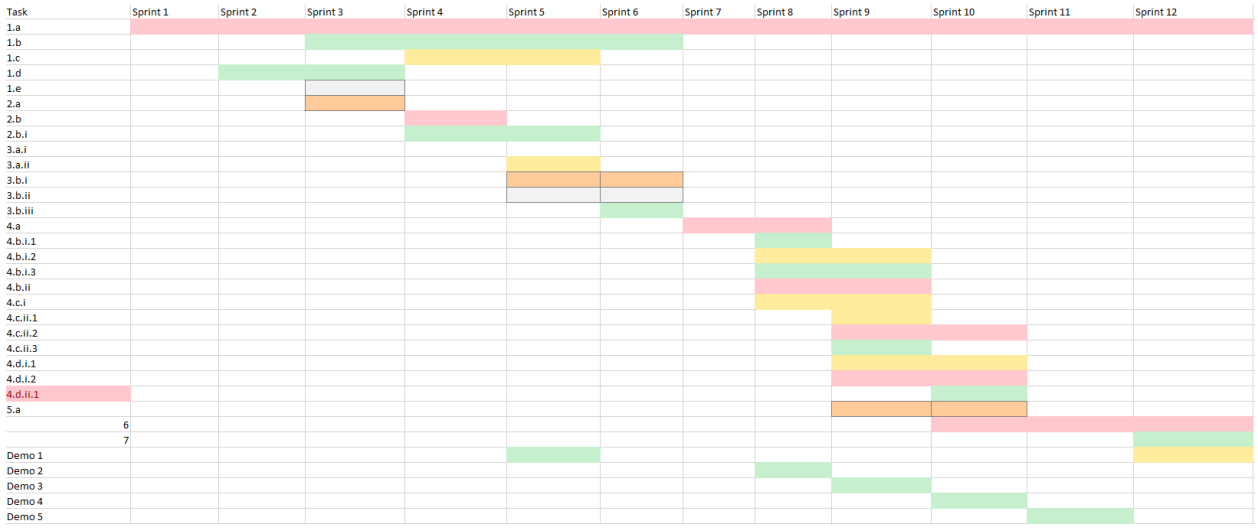


Figure 1. Project Gantt Chart

2.5 Project Tracking Procedures

This project will be tracked in many ways that will allow us to properly manage project completion and assure that requirements are being properly met. This project will mainly be tracked via use of github repositories and trello tasks. As a course, this class will be tracked via the regular submission of deliverables and design documents. Additionally, continually monitoring the project and discussing project standards and updates will allow us to continually make changes to ensure a more successful end project.

2.6 Personnel Effort Requirements

This project will require each member of the team to be personally responsible for their share of the project. Individual achievements and progress will be tracked easily through our choice of project management tools as each will be linked to one, or many team members. However, much of this project relies on each individual being responsible for completing a significant portion of the project. This is a 3 credit course, and the second semester course is 2 credit hours. This project should amount for roughly 5 credit hours worth of work for each individual within the group. However, this will vary widely for each individual project.

2.7 Other Resource Requirements

Through discussions with the client responsible for sponsoring this project, many of the physical requirements for the project are already in place at their facility and this code would take advantage of already in place physical systems. Other open source resources are being taken advantage of as parts of this project as well in order to complete the project. These resources include video conferencing tools such as Zoom and Webex, team discussion tools such as Gmail and GroupMe, project management tools such as Trello and Github, and likely many more resources that have not been needed as of yet. These resources will be properly credited and are an essential part of completing our project successfully.

2.8 Financial Requirements

This project is sponsored under Collins Aerospace as a professional development project development experience. However, cost and financial requirements are going to be heavily accessed within this project. As a software development project, this will have very limited financial requirements. Additionally, many of the financial requirements of this project have already been met as Collins has a dedicated system already in use and will likely not need further physical resources to implement. Much of the potential cost of a project like this is associated with manhours, salary of workers, and potential physical development needed to support code. This project lacks many of these costs as we will all be completing this project as a part of a course, and the physical requirements are already in place.

3 Design

3.1 Previous Work and Literature

An Engineer at Collins has developed a simple code generator. It is limited to code generation in Ada, and does not work for all use cases. It does not generate all files required. It is also temperamental when used. There are code generation tools and XML/Ada code examples.

Other open-source generators were surveyed for applicability to the project. The OpenAPI generator, Yeoman, hygen, and JHipster were surveyed. Amongst these, the OpenAPI generator most closely matched our use case. We used this as a pattern to base our generator off of.

3.2 Design Thinking

Collins has specified that they want a code generator, which eliminates several options to solve the problem. Within the code generation field, there are several options. The current generator creates a file using a formatted string. While this works, it is difficult to maintain. Thus, we are looking at using templating engines to create our files. We are also looking into options for xml file validation.

3.3 Proposed Design

The proposed design consists of seven primary modules. The first is the core module. The core module is responsible for parsing the program arguments and passing them the necessary module. The second module is Validation. This module is responsible for validating the XMLl file against the XML schema. It is also responsible for throwing verbose and useful error messages regarding file validation errors. The next module is the configuration module. This module is responsible for adding language and message specific settings for the generation. This is then passed to the Generation module. The generation module is responsible for getting the raw data from the parsing module, putting the data in the proper format, and then passing it to the templating module. The parsing module reads the XML file, and puts the data into java objects. The templating module takes the processed data and produces the contents of a file using a templating engine. It is also responsible for writing the file to disk. For this project, we plan on using Handlebars for templating . The common module is a simple module containing resources needed by multiple modules. The final module is the CLI. It would be responsible for sending arguments to the core module. Having a separate CLI modular allows for the possibility of other interfaces in the future with minimum refactoring.

A standard generation workflow would look something like this. A user would start a generation with the CLI. The core module would use the validation module to validate the input file. After validation, the configuration module would provide a configuration based upon the program arguments. This would be passed to the generation module. The generation module would then call the parsing module to read in the XML file. The generation module would then process the data, then pass it to the templating module. The templating module would then create the file contents using Mustache, then write it to file.

This was designed largely with maintainability and extensibility in mind. There are many ways to produce a file, but a major requirement of this project is to be extensible as needs change in the future. The client has stated that the code being generated is guaranteed to change over time. The breakdown of modules allows the project to be easily extended in the future.

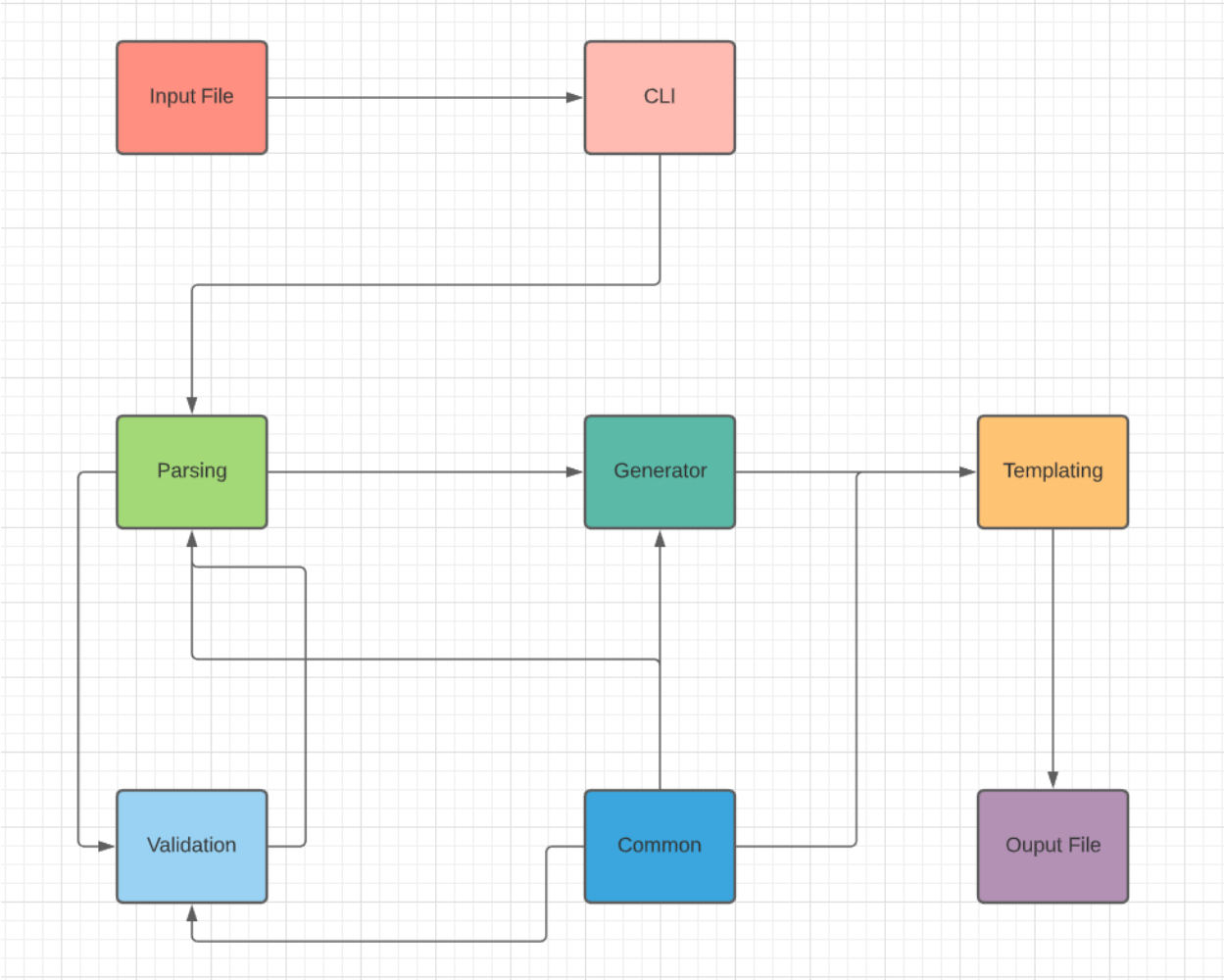


Figure 2. Module Diagram

3.4 Technology Considerations

We will be using java as the primary language. This allows the project to be easily maintained, due the prolific knowledge of java. One drawback of this is that the project, while written in java, will also require a knowledge of Ada.

3.5 Design Analysis

A simple prototype of the project was created for a demo to the client. While nowhere near complete, it was properly able demonstrate most of the major functionalities of the project. Based on this, we expect the design to work.

We expect that use of an Agile development will help with design iterations. With regular demos and meetings with the client, if the design does not match expectation, this can be identified rapidly, allowing the design to be changed with minimum lost effort. Requirements clarifications will also allow use to update the design as issues become apparent.

3.6 Development Process

For this project, we plan on using Agile development processes. We plan on having stand up meetings, sprints, and weekly meetings with the Client. This fits well with the clients work processes, and fits in well with the COVID workflow for college students. We are using Gitlab and will be using merge branch workflow.

3.7 Design Plan

For this project, our design plan is shaped through our client's intended work concerning their development staff and testing engineers. In regards to requirements, the plan is to sequentially instantiate the modules through calculated optimistic and pessimistic outcomes set by time estimates controlled by the preceding task. Below is our block diagram representation of the design plan as deconstructed via modules.

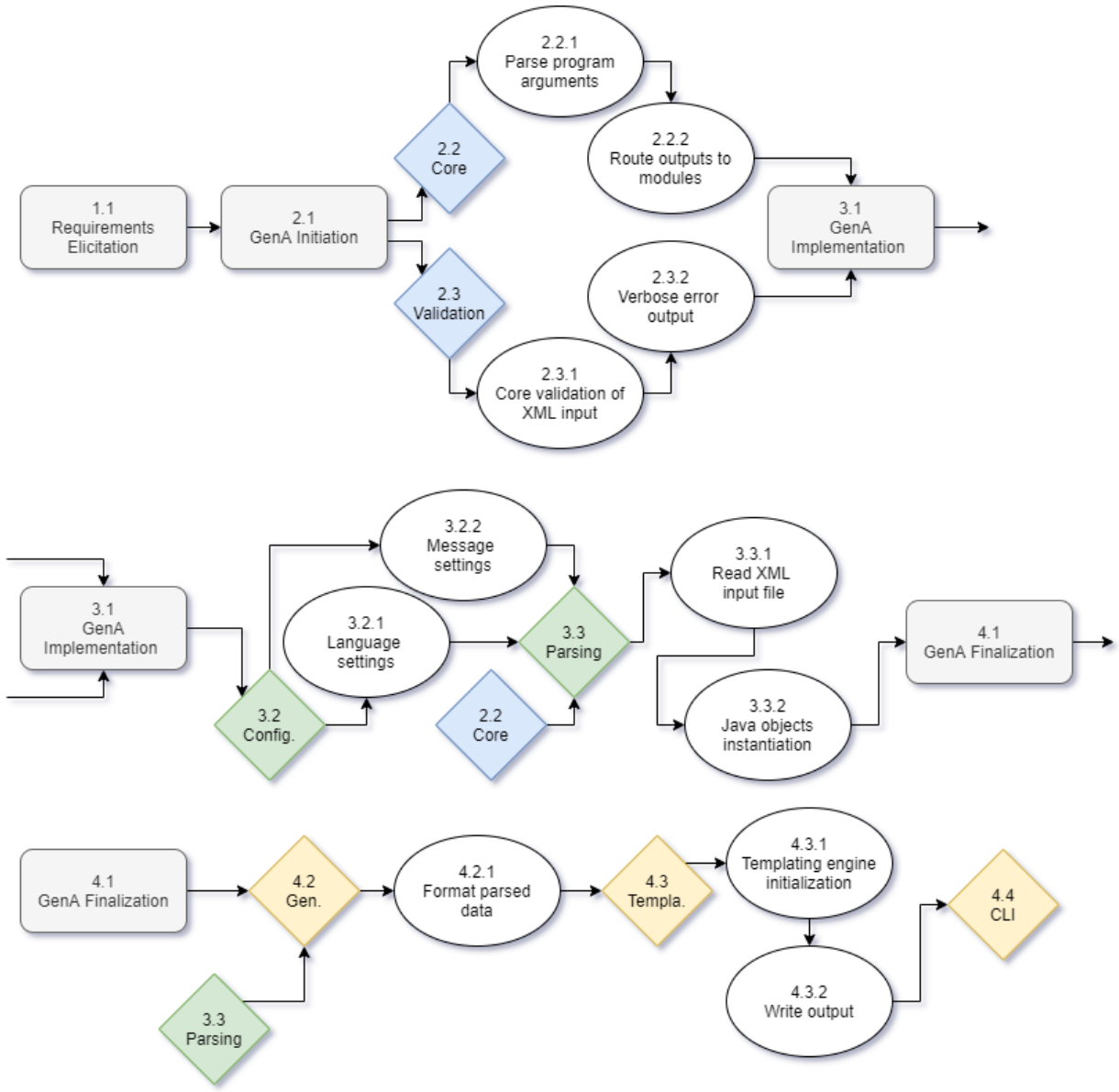


Figure 3. Design Plan

4 Testing

4.1 Unit Testing

4.2 Interface Testing

The interface between the UI and the Java modules will be tested. Interfaces will be the CLI interface to be run in the Windows CMD environment. Time permitting, a GUI will be developed,

which will also be tested. GUI testing will be done with Selenium. We will also implement an integration testing suite to allow us to confirm the functionality of the module interfaces.

4.3

Acceptance testing provides some challenges. Due to security concerns, the team will not be able to test our generated code in its final environment. Communication with the client will be critical, both for ensuring the generator functions as the client expects and ensuring the generated code functions properly. Confirming that the generated code can compile under Ada is a form of acceptance testing that the team can perform themselves. As discussed with the client, acceptance testing will likely include sending generated code to the client for testing. The client will then perform testing on the code.

4.4 Results

Minimal testing has been performed as part of a demo. Testing is a large part of the development of this project outlined for the second semester of this course set, and will produce results as development further progresses.

5 Implementation

For implementation, we plan on having two week sprints. Given the shortened Spring semester, we anticipate 6 sprints over a 12 week period. This should provide sufficient time to implement the project. We intend to follow the plan laid out in the gantt chart.

6 Closing Material

6.1 Conclusion

In this project, many plans and baseline development has been completed to provide the best possible base for the development of our project throughout the rest of this semester and in the next semester. We have laid a sustainable task management system, group discussion processes and schedules, and have deeply discussed the requirements and goals of this project.

GenA, the proposed project in conjunction with Collins Aerospace, proposes using NDOs for use in avionics systems as a means of intercommunication between systems. This must be completed through a strict schema already used within this system to send messages between system subsets. This schema is then used to implement the message in Ada. However, because of the drawbacks of this current system such as errors, time consumption, and correction effort, Collins Aerospace wishes to produce NDOs more efficiently and effectively.

The solution to this problem proposed by our team will implement a code generator to create NDO messages from the XML schema provided using the XML file already being produced during the Collins Aerospace currently in use processes. This will then create the ADA file directly, reducing the potential for errors, and providing a more efficient, and understandable process to work with. This solution will greatly improve on the current system, and produce a much more user friendly, and efficient experience.

6.2 References

N/A

6.3 Appendices

There are no applicable appendices at this time.

5.3.3 Appendix III: Other Considerations

N/A

5.3.4 Appendix IV: Relevant Code

This appendix will show the input XML and generated ADA code. The data in the input file is processed, and template 1 is used to create Ada file 1. Template 2 is used to create Ada file 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<CatalogInterface Name="Guidance" Version="Version 10" Direction="Output"
InterfaceType="UDP">
   <InterfaceConfig>
      <UDPConfig WKS="Guidance" InterfaceType="Parametric"/>
   </InterfaceConfig>
   <NDOMessage Name="First_Ndo" ID="3550001" Length="26" Rate="20"
Format="NORMAL_FORMAT">
      <DataElement Name="First_Float" ByteOffset="0" DataSize="4"
SizeIn="Bytes" DataType="IEEE Float" Units="Percent"
                   ElementReference="Percent_Type"/>
      <DataElement Name="Second_Float" ByteOffset="4" DataSize="4"
SizeIn="Bytes" DataType="IEEE Float"
                   Units="Percent" ElementReference="Percent_Type"/>
      <DataElement Name="Third_Float" ByteOffset="8" DataSize="4"
SizeIn="Bytes" DataType="IEEE Float" Units="Percent"
                   ElementReference="Percent_Type"/>
      <DataElement Name="First_Boolean" ByteOffset="12" DataSize="1"
SizeIn="Bytes" DataType="Boolean"/>
      <DataElement Name="Second_Boolean" ByteOffset="13" DataSize="1"
SizeIn="Bytes" DataType="Boolean"/>
      <DataElement Name="First_Signed_int" ByteOffset="14" DataSize="4"
SizeIn="Bytes" DataType="SignedInt"
                   ElementReference="Signed_Int_Range_Type"/>
      <DataElement Name="Second_Signed_Int" ByteOffset="18" DataSize="4"
SizeIn="Bytes" DataType="SignedInt"/>
      <DataElement Name="Spare2" ByteOffset="22" DataSize="1" SizeIn="Bytes"
DataType="Spare"/>
      <DataElement Name="Spare3" ByteOffset="23" DataSize="1" SizeIn="Bytes"
DataType="Spare"/>
      <DataElement Name="Spare4" ByteOffset="24" DataSize="1" SizeIn="Bytes"
DataType="Spare"/>
      <DataElement Name="Modes" ByteOffset="25" DataSize="1" SizeIn="Bytes"
DataType="Enumeration"
                   ElementReference="Modes_Type"/>
      <Range Name="Percent_Type">
         <RangeElement Name="Valid_Range" LowerBound="-100"
UpperBound="100"/>
      </Range>
      <Range Name="Signed_Int_Range_Type">
         <RangeElement LowerBound="-10" UpperBound="10"/>
      </Range>
      <Enumeration Name="Modes_Type">
         <EnumElement Name="Secondary" Default="0"/>
         <EnumElement Name="Primary" Default="1"/>
      </Enumeration>
   </NDOMessage>
   <NDOMessage Name="Second_Ndo" ID="3550002" Length="18" Rate="20"
Format="NORMAL_FORMAT">
      <DataElement Name="Speed" ByteOffset="0" DataSize="4" SizeIn="Bytes"
DataType="IEEE Float" Units="knots"/>
      <DataElement Name="Altitude" ByteOffset="4" DataSize="4" SizeIn="Bytes"
```

```
DataType="IEEE Float"/>
       <DataElement Name="Baro_Set" ByteOffset="8" DataSize="4" SizeIn="Bytes"
DataType="UnsignedInt"
                    ElementReference="Baro_Set_Range"/>
       <DataElement Name="Type_Of_Speed" ByteOffset="12" DataSize="1"
SizeIn="Bytes" DataType="Enumeration"
                    ElementReference="Speed_Type"/>
       <DataElement Name="Type_Of_Alitutde" ByteOffset="13" DataSize="1"
SizeIn="Bytes" DataType="Enumeration"
                    ElementReference="Altitude_Type"/>
       <DataElement Name="Speed_Valid" ByteOffset="14" DataSize="1"
SizeIn="Bytes" DataType="Boolean"/>
       <DataElement Name="Altitude_Valid" ByteOffset="15" DataSize="1"
SizeIn="Bytes" DataType="Boolean"/>
       <DataElement Name="Baroset_valid" ByteOffset="16" DataSize="1"
SizeIn="Bytes" DataType="Boolean"/>
       <DataElement Name="Spare1" ByteOffset="17" DataSize="1" SizeIn="Bytes"
DataType="Spare"/>
       <Range Name="Baro_Set_Range">
           <RangeElement LowerBound="0" UpperBound="31"/>
       </Range>
       <Enumeration Name="Speed_Type">
           <EnumElement Name="Airspeed" Default="0"/>
           <EnumElement Name="Groundspeed" Default="2"/>
           <EnumElement Name="Warp_Speed" Default="3"/>
       </Enumeration>
       <Enumeration Name="Altitude_Type">
           <EnumElement Name="Invalid" Default="0"/>
           <EnumElement Name="Baro" Default="1"/>
           <EnumElement Name="Rad" Default="2"/>
       </Enumeration>
   </NDOMessage>
</CatalogInterface>
```

Fig. 2 Input XML

```
-- Copyright 2020 Rockwell Collins.  all Rights Reserved.--
-- PACKAGE NAME: example_code
--
-- FILE NAME: example_code.1.ada
--
-- AUTHOR:
--
-- DATE:
--
-- PURPOSE:
--
-- REVISION HISTORY:
--
-- REVISION:
-- DATE:
-- AUTHOR:
```

```
-- DESCRIPTION:
-- REFERENCE:
--
--
-- EXTERNAL ACCESS:
--
with Interfaces;
use Interfaces;
with Common_Io.Byte_Order;
with Config_Types;
package Example_Code is


-- This type definition is required to instantiate the LAN generics.
-- It should have one enumeration for each NDO.
type {{name}} is
({{#each nDOMessage}}{{#unless @last}}{{name}},{{/unless}}{{#if
@last}}{{name}}{{/if}}{{/each}});

-- WKN/WKS for LAN
{{name}}_Wkn : constant String := "{{name}}";
{{name}}_Wks : constant String := "{{name}}";

{{#each nDOMessage}}
   {{#each range}}
       type {{name}} is new {{dataType}} range {{rangeElement.lowerBound}} ..
{{rangeElement.upperBound}};
       for {{name}}'Size use {{dataLength}};
   {{/each}}

   {{#each enumeration}}
       type {{name}} is ({{#each enumElement}}{{#unless
@last}}{{name}},{{/unless}}{{#if
                @last}}{{name}}{{/if}}{{/each}});
       for {{name}}'Size use ({{#each enumElement}}{{#unless @last}}{{name}}
=> {{default}},{{/unless}}{{#if
                @last}}{{name}}=> {{default}}{{/if}}{{/each}});
       for {{name}}'Size use {{dataLength}};
   {{/each}}

   type {{name}}_Type is
   record
   {{#each dataElement}}{{name}}    :   {{elementReference}};
   {{/each}}end record;

   for {{name}}_Type use
   record
   {{#each dataElement}}{{name}}  at 0 range {{startBit}} .. {{endBit}};
   {{/each}}end record;
   for {{name}}_Type'Size use {{bitLength}};

   type {{name}}_Type_Format_Type is
   record
   Record_Def        : Common_Io.Byte_Order.Data_Description;
   Element_Count     : Common_Io.Byte_Order.Array_Or_Record_Length;
   {{#each dataElement}}{{name}}    :   Common_Io.Byte_Order.Data_Description;
```

```
   {{/each}}
   end record;

   for {{name}}_Type_Format_Type use
   record
   Record_Def at 0 range 0 .. 15;
   Element_Count at 0 range 16 .. 31;
   {{#each dataElement}}{{name}}  at 0 range {{adaTypeFormatTypeStartBit}} ..
{{adaTypeFormatTypeEndBit}};
   {{/each}}end record;
   for {{name}}_Type_Format_Type'Size use {{typeFormatTypeLength}};

   {{name}}_Type_Format : constant {{name}}_Type_Format_Type :=
   (Record_Def        => Common_Io.Byte_Order.Is_Record,
   Element_Count     => {{numElements}},
   {{#each dataElement}}{{#unless @last}}{{name}} =>
Common_Io.Byte_Order.Is_{{bitLength}}_Bits,
   {{/unless}}{{#if  @last}}{{name}} =>
Common_Io.Byte_Order.Is_{{bitLength}}_Bits{{/if}}{{/each}});

{{/each}}

{{#each nDOMessage}}
   procedure Send_Data (Data : {{name}}_Type);
{{/each}}

Shift_8_Bits            : constant Config_Types.Data_Id_Type := 256;

{{#each nDOMessage}}
   {{name}}_Id             : constant Config_Types.Data_Id_Type :=
{{iD}};{{/each}}

{{name}}_{{interfaceConfig.uDPConfig.interfaceType}}_Config
constant array ({{name}}_Data_Type) of
Config_Types.{{interfaceConfig.uDPConfig.interfaceType}}_Config_Type :=
({{#each nDOMessage}}{{#unless @last}}{{name}}   => (Id => {{name}}_Id *
Shift_8_Bits + {{@key}} + 1,
Size_In_Bytes    => First_Ndo_Type'Size / 8,
Output_Time_Ms   => 50,
Stale_Timeout_Ms => 150,
Data_Definition  => {{name}}_Type_Format'Address),
{{/unless}}
{{#if @last}}{{name}}  => (Id => {{name}}_Id * Shift_8_Bits + {{@key}} +1,
Size_In_Bytes    => First_Ndo_Type'Size / 8,
Output_Time_Ms   => 50,
Stale_Timeout_Ms => 150,
Data_Definition  => {{name}}_Type_Format'Address));
{{/if}}
{{/each}}

end Example_Code;
```

Fig 3. Template 1

```
-- Copyright 2020 Rockwell Collins.  all Rights Reserved.--
-- PACKAGE NAME: example_code
--
-- FILE NAME: example_code.1.ada
--
-- AUTHOR:
--
-- DATE:
--
-- PURPOSE:
--
-- REVISION HISTORY:
--
-- REVISION:
-- DATE:
-- AUTHOR:
-- DESCRIPTION:
-- REFERENCE:
--
--
-- EXTERNAL ACCESS:
--
with Interfaces;
use Interfaces;
with Common_Io.Byte_Order;
with Config_Types;
package Example_Code is


  -- This type definition is required to instantiate the LAN generics.
  -- It should have one enumeration for each NDO.
  type Guidance_Data_Type is
    (First_Ndo, Second_Ndo);

  -- WKN/WKS for LAN
  Guidance_Wkn : constant String := "Guidance";
  Guidance_Wks : constant String := "Guidance";

  type Percent_Type is new IEEE_Float_32 range -100.0 .. 100.0;
  for Percent_Type'Size use 32;

  type Signed_Int_Range_Type is new Integer range -10 .. 10;
  for Signed_Int_Range_Type'Size use 32;

  type Modes_Type is (Secondary, Primary);
  for Modes_Type use (Secondary => 1,
                      Primary   => 2);
  for Modes_Type'Size use 8;

  type First_Ndo_Type is
     record
        First_Float       : Percent_Type;
        Second_Float      : Percent_Type;
        Third_Float       : Percent_Type;
        First_Boolean     : Boolean;
        Second_Boolean    : Boolean;
```

```
      First_Signed_Int  : Signed_Int_Range_Type;
      Second_Signed_Int : Integer;
      Spare2            : Boolean;
      Spare3            : Boolean;
      Spare4            : Boolean;
      Modes             : Modes_Type;
   end record;

for First_Ndo_Type use
   record
      First_Float at 0 range 0 .. 31;  -- 32
      Second_Float at 0 range 32 .. 63;  -- 32
      Third_Float at 0 range 64 .. 95;  -- 32
      First_Boolean at 0 range 96 .. 103;  -- 8
      Second_Boolean at 0 range 104 .. 111;  -- 8
      First_Signed_Int at 0 range 112 .. 143;  -- 32
      Second_Signed_Int at 0 range 144 .. 175;  -- 32
      Spare2 at 0 range 176 .. 183;  -- 8
      Spare3 at 0 range 184 .. 191;  -- 8
      Spare4 at 0 range 192 .. 199;  -- 8
      Modes at 0 range 200 .. 207;  -- 8
   end record;            -- Record size is 208 bits
for First_Ndo_Type'Size use 208;


type First_Ndo_Type_Format_Type is
   record
      Record_Def        : Common_Io.Byte_Order.Data_Description;
      Element_Count     : Common_Io.Byte_Order.Array_Or_Record_Length;
      First_Float       : Common_Io.Byte_Order.Data_Description;
      Second_Float      : Common_Io.Byte_Order.Data_Description;
      Third_Float       : Common_Io.Byte_Order.Data_Description;
      First_Boolean     : Common_Io.Byte_Order.Data_Description;
      Second_Boolean    : Common_Io.Byte_Order.Data_Description;
      First_Signed_Int  : Common_Io.Byte_Order.Data_Description;
      Second_Signed_Int : Common_Io.Byte_Order.Data_Description;
      Spare2            : Common_Io.Byte_Order.Data_Description;
      Spare3            : Common_Io.Byte_Order.Data_Description;
      Spare4            : Common_Io.Byte_Order.Data_Description;
      Modes             : Common_Io.Byte_Order.Data_Description;
   end record;


for First_Ndo_Type_Format_Type use
   record
      Record_Def at 0 range 0 .. 15;
      Element_Count at 0 range 16 .. 31;
      First_Float at 0 range 32 .. 47;
      Second_Float at 0 range 48 .. 63;
      Third_Float at 0 range 64 .. 79;
      First_Boolean at 0 range 80 .. 95;
      Second_Boolean at 0 range 96 .. 111;
      First_Signed_Int at 0 range 112 .. 127;
      Second_Signed_Int at 0 range 128 .. 143;
      Spare2 at 0 range 144 .. 159;
      Spare3 at 0 range 160 .. 175;
```

```
      Spare4 at 0 range 176 .. 191;
      Modes at 0 range 192 .. 207;
   end record;             -- Record size is 208 bits
  for First_Ndo_Type_Format_Type'Size use 208;


  First_Ndo_Type_Format : constant First_Ndo_Type_Format_Type :=
                          (Record_Def        =>
Common_Io.Byte_Order.Is_Record,
                           Element_Count     => 11,
                           First_Float       =>
Common_Io.Byte_Order.Is_32_Bits,
                           Second_Float      =>
Common_Io.Byte_Order.Is_32_Bits,
                           Third_Float       =>
Common_Io.Byte_Order.Is_32_Bits,
                           First_Boolean     =>
Common_Io.Byte_Order.Is_8_Bits,
                           Second_Boolean    =>
Common_Io.Byte_Order.Is_8_Bits,
                           First_Signed_Int  =>
Common_Io.Byte_Order.Is_32_Bits,
                           Second_Signed_Int =>
Common_Io.Byte_Order.Is_32_Bits,
                           Spare2            =>
Common_Io.Byte_Order.Is_8_Bits,
                           Spare3            =>
Common_Io.Byte_Order.Is_8_Bits,
                           Spare4            =>
Common_Io.Byte_Order.Is_8_Bits,
                           Modes             =>
Common_Io.Byte_Order.Is_8_Bits);

  type Baro_Set_Range is new IEEE_Float_32 range 0.0 .. 31.0;
  for Baro_Set_Range'Size use 32;

  type  Speed_Type is (Airspeed, Groundspeed, Warp_Speed);
  for Speed_Type use (Airspeed => 0, Groundspeed => 2, Warp_Speed => 3);
  for Speed_Type'Size use 8;

  type  Altitude_Type is (Invald, Baro, Rad);
  for Altitude_Type use (Invald => 0, Baro => 1, Rad => 2);
  for Altitude_Type'Size use 8;

  type Second_Ndo_Type is
     record
        Speed                     : Ieee_Float_32;
        Altitude                  : Ieee_Float_32;
        Baro_Set                  : Baro_Set_Range;
        Type_Of_Speed             : Speed_Type;
        Type_Of_Alitude           : Altitude_Type;
        Speed_Valid               : Boolean;
        Altitude_Valid            : Boolean;
        Baroset_Valid             : Boolean;
        Spare1                    : Boolean;
     end record;
```

```
  for Second_Ndo_Type use
     record
        Speed at 0 range 0 .. 31;  -- 32
        Altitude at 0 range 32 .. 63;  -- 32
        Baro_Set at 0 range 64 .. 95;  -- 32
        Type_Of_Speed at 0 range 96 .. 103;  -- 8
        Type_Of_Alitude at 0 range 104 .. 111;  -- 8
        Speed_Valid at 0 range 112 .. 119;  -- 8
        Altitude_Valid at 0 range 120 .. 127;  -- 8
        Baroset_Valid at 0 range 128 .. 135;  -- 8
        Spare1 at 0 range 136 .. 143;  -- 8
     end record;             -- Record size is 144 bits
  for Second_Ndo_Type'Size use 144;


  type Second_Ndo_Type_Format_Type is
     record
        Record_Def      : Common_Io.Byte_Order.Data_Description;
        Element_Count   : Common_Io.Byte_Order.Array_Or_Record_Length;
        Speed           : Common_Io.Byte_Order.Data_Description;
        Altitude        : Common_Io.Byte_Order.Data_Description;
        Baro_Set        : Common_Io.Byte_Order.Data_Description;
        Type_Of_Speed   : Common_Io.Byte_Order.Data_Description;
        Type_Of_Alitude : Common_Io.Byte_Order.Data_Description;
        Speed_Valid     : Common_Io.Byte_Order.Data_Description;
        Altitude_Valid  : Common_Io.Byte_Order.Data_Description;
        Baroset_Valid   : Common_Io.Byte_Order.Data_Description;
        Spare1          : Common_Io.Byte_Order.Data_Description;
     end record;


  for Second_Ndo_Type_Format_Type use
     record
        Record_Def at 0 range 0 .. 15;
        Element_Count at 0 range 16 .. 31;
        Speed at 0 range 32 .. 47;
        Altitude at 0 range 48 .. 63;
        Baro_Set at 0 range 64 .. 79;
        Type_Of_Speed at 0 range 80 .. 95;
        Type_Of_Alitude at 0 range 96 .. 111;
        Speed_Valid at 0 range 112 .. 127;
        Altitude_Valid at 0 range 128 .. 143;
        Baroset_Valid at 0 range 144 .. 159;
        Spare1 at 0 range 160 .. 175;
     end record;             -- Record size is 176 bits
  for Second_Ndo_Type_Format_Type'Size use 176;


  Second_Ndo_Type_Format : constant Second_Ndo_Type_Format_Type :=
                           (Record_Def      =>
Common_Io.Byte_Order.Is_Record,
                            Element_Count   => 9,
                            Speed           =>
Common_Io.Byte_Order.Is_32_Bits,
                            Altitude        =>
```

```
Common_Io.Byte_Order.Is_32_Bits,
                              Baro_Set         =>
Common_Io.Byte_Order.Is_32_Bits,
                              Type_Of_Speed    =>
Common_Io.Byte_Order.Is_8_Bits,
                              Type_Of_Alitude =>
Common_Io.Byte_Order.Is_8_Bits,
                              Speed_Valid      =>
Common_Io.Byte_Order.Is_8_Bits,
                              Altitude_Valid   =>
Common_Io.Byte_Order.Is_8_Bits,
                              Baroset_Valid    =>
Common_Io.Byte_Order.Is_8_Bits,
                              Spare1           =>
Common_Io.Byte_Order.Is_8_Bits);
  procedure Send_Data (Data : First_Ndo_Type);
  -- Needed to shift the NDO ID into the upper part of the ID field.
  Shift_8_Bits           : constant Config_Types.Data_Id_Type := 256;

  -- NDO ID
  First_Ndo_Id             : constant Config_Types.Data_Id_Type := 355001;
  Second_Ndo_Id            : constant Config_Types.Data_Id_Type := 355002;


  -- Configuration structure used to register the NDO
  Guidance_Periodic_Config          :
  constant array (Guidance_Data_Type) of
    Config_Types.Periodic_Config_Type :=
      (First_Ndo               => (Id => First_Ndo_Id * Shift_8_Bits + 1,
                                   Size_In_Bytes    => First_Ndo_Type'Size /
8,
                                   Output_Time_Ms   => 50,
                                   Stale_Timeout_Ms => 150,
                                   Data_Definition  =>
First_Ndo_Type_Format'Address),

       Second_Ndo              => (Id               => Second_Ndo_ID *
Shift_8_Bits + 2,
                                   Size_In_Bytes    => Second_Ndo'Size / 8,
                                   Output_Time_Ms   => 50,
                                   Stale_Timeout_Ms => 150,
                                   Data_Definition  =>
Second_Ndo_Type_Format'Address));

end Example_Code;
```

Fig 4. Ada Output File 1

```
-- Copyright 2020 Rockwell Collins.  All Rights Reserved.
--
-- PACKAGE NAME: example_code
--
-- FILE NAME: example_code.2.ada
--
-- AUTHOR:
--
-- DATE:
--
-- PURPOSE:
--
-- REVISION HISTORY:
--
-- REVISION:
-- DATE:
-- AUTHOR:
-- DESCRIPTION:
-- REFERENCE:
--
--
-- EXTERNAL ACCESS:
with Periodic_Transmit;
pragma Elaborate (Periodic_Transmit);
package body  Example_Code is

  -- GENERIC INSTANTIATIONS
  package Guidance_Tx is
    new Periodic_Transmit (Transmit_Type             => Guidance_Data_Type,
                           Well_Known_Name           => Guidance_Wkn,
                           Well_Known_Service        => Guidance_Wks,
                           Transmitting_Arinc_Labels => False);




  --.....................................................................
  --
  -- FUNCTION AND/OR PROCEDURE HEADER BLOCKS

  -- NAME: Send_Data
  --
  -- PURPOSE: Send provided data on the network
  --
  -- EXCEPTIONS/ERROR CONDITIONS:
  --
  --
  procedure Send_Data (Data : First_Ndo_Type) is

     Local_Data : First_Ndo_Type;

  begin

     Local_Data := Data;

     Guidance_Tx.Transmit (Data_Type     => First_Ndo,
```

```
                            Data_Location => Local_Data'Address);


   end Send_Data;

   --.....................................................................
   --
   -- FUNCTION AND/OR PROCEDURE HEADER BLOCKS

   -- NAME: Send_Data
   --
   -- PURPOSE: Send provided data on the network
   --
   -- EXCEPTIONS/ERROR CONDITIONS:
   --
   --
   procedure Send_Data (Data : Second_Ndo_Type) is

      Local_Data : Second_Ndo_Type;

   begin

      Local_Data := Data;

      Guidance_Tx.Transmit (Data_Type     => Second_Ndo,
                            Data_Location => Local_Data'Address);


   end Send_Data;
   --.....................................................................
   --
   -- FUNCTION AND/OR PROCEDURE HEADER BLOCKS

   -- NAME: Initialize
   --
   -- PURPOSE: Initializes the queue for data transmission.
   --
   -- EXCEPTIONS/ERROR CONDITIONS:
   --
   --
   procedure Initialize is
   begin

      Guidance_Tx.Initialize (Config =>
                                 Guidance_Tx.Config_Type
                                   (Guidance_Periodic_Config));

   end Initialize;

end Example_Code;
```

Fig 5. Template 2

```
-- Copyright 2020 Rockwell Collins.  All Rights Reserved.
--
-- PACKAGE NAME: example_code
--
-- FILE NAME: example_code.2.ada
--
-- AUTHOR:
--
-- DATE:
--
-- PURPOSE:
--
-- REVISION HISTORY:
--
-- REVISION:
-- DATE:
-- AUTHOR:
-- DESCRIPTION:
-- REFERENCE:
--
--
-- EXTERNAL ACCESS:
with Periodic_Transmit;
pragma Elaborate (Periodic_Transmit);
package body  Example_Code is

  -- GENERIC INSTANTIATIONS
  package Guidance_Tx is
    new Periodic_Transmit (Transmit_Type             => Guidance_Data_Type,
                           Well_Known_Name           => Guidance_Wkn,
                           Well_Known_Service        => Guidance_Wks,
                           Transmitting_Arinc_Labels => False);




  --.....................................................................
  --
  -- FUNCTION AND/OR PROCEDURE HEADER BLOCKS

  -- NAME: Send_Data
  --
  -- PURPOSE: Send provided data on the network
  --
  -- EXCEPTIONS/ERROR CONDITIONS:
  --
  --
  procedure Send_Data (Data : First_Ndo_Type) is

     Local_Data : First_Ndo_Type;

  begin

     Local_Data := Data;

     Guidance_Tx.Transmit (Data_Type     => First_Ndo,
```

```
                            Data_Location => Local_Data'Address);


  end Send_Data;

  --.....................................................................
  --
  -- FUNCTION AND/OR PROCEDURE HEADER BLOCKS

  -- NAME: Send_Data
  --
  -- PURPOSE: Send provided data on the network
  --
  -- EXCEPTIONS/ERROR CONDITIONS:
  --
  --
  procedure Send_Data (Data : Second_Ndo_Type) is

     Local_Data : Second_Ndo_Type;

  begin

     Local_Data := Data;

     Guidance_Tx.Transmit (Data_Type     => Second_Ndo,
                           Data_Location => Local_Data'Address);


  end Send_Data;
  --.....................................................................
  --
  -- FUNCTION AND/OR PROCEDURE HEADER BLOCKS

  -- NAME: Initialize
  --
  -- PURPOSE: Initializes the queue for data transmission.
  --
  -- EXCEPTIONS/ERROR CONDITIONS:
  --
  --
  procedure Initialize is
  begin

     Guidance_Tx.Initialize (Config =>
                               Guidance_Tx.Config_Type
                                 (Guidance_Periodic_Config));

  end Initialize;

end Example_Code;
```

Fig 6. Ada Output File 2