NASA/TM—2020-220513

On Development of Modern Software Interface to Glenn Research Center's Communication Analysis Suite

Hady Salama and Bryan W. Welch
Glenn Research Center, Cleveland, Ohio

June 2020

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers, but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., "quick-release" reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 757-864-6500

- Telephone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Program
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM—2020-220513

On Development of Modern Software Interface to Glenn Research Center's Communication Analysis Suite

Hady Salama and Bryan W. Welch
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

June 2020

This report contains preliminary findings, subject to revision as analysis proceeds.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

This report is available in electronic form at http://www.sti.nasa.gov/ and http://ntrs.nasa.gov/

On Development of Modern Software Interface to Glenn Research Center's Communication Analysis Suite

Hady Salama* and Bryan W. Welch
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Summary

NASA's Space Communications and Navigation (SCaN) program analyzes space communication channels involving satellites in any Earth orbit and deep space for a multitude of operations. Specifically, the SCaN program will power the future of lunar communications in the next decade and beyond. A web-based user interface of the static link analysis component of the Glenn Research Center's Communication Analysis Suite (GCAS) was developed. It is operated through a newly designed front-end application with an easy to use and intuitive web-based user interface that provides accurate satellite communications link analysis capabilities. Its use does not require any knowledge of the underlying MATLAB® program (The MathWorks, Inc.) or its corresponding programming environment. The GCAS offers high-fidelity optical communications and radiofrequency link analysis calculations to give users the ability to input a multitude of known link parameters and calculate desired link analysis outputs on a new web-based software platform. This interface utilizes significantly more link parameters in the calculations, compared with prior versions of the static link analysis capability, with the ability to solve for a larger set of output results in different configuration modes, and fidelity and added flexibility in terms of developing software outside of MATLAB®. Features included in the web application utilize the MATLAB® Runtime™ to perform calculations using the existing repository of MATLAB® code, save configuration parameter values, load configuration parameter values with preloaded default parameter values, to increase the tool's versatility. The updates to the GCAS interface will benefit NASA as it develops the next decade's lunar satellite communication architecture. A broader population of users will have access to modernized communications link analysis as more advanced satellite communications architectures are built.

Introduction

NASA's Space Communications and Navigation (SCaN) program analyzes and operates satellite and ground station systems to enable critical space communications for NASA missions. These missions and operations are powered by radiofrequency (RF) communications links today, and may in the future include optical (laser beam) communications links. User satellites establish effective communications links with other satellites (space-to-space links) and/or with ground station antennas (space-to-ground or ground-to-space links). The performance of the links from space-to-space, space-to-ground, or ground-to-space are characterized by the quality and timing of the data received. The analysis of the links are affected by over 60 variables including the distance between the nodes, transmit power, receiver gain, data rate, modulation, coding scheme, and lens diameter in optical links. Solving for variables such as link margin, distance, data rate, and transmitter power are critical when analyzing the effectiveness of a

*Summer Intern in Lewis' Educational and Research Collaborative Internship Program (LERCIP), undergraduate at Ohio State University.

communications link. Since the mathematical models that solve for such values must be used frequently in any analysis of a space mission, it is important to develop efficient software using these models to quickly solve for a desired output. It is also important to build an intuitive and easy-to-operate graphical user interface (GUI) to maximize the use of the tool for those not familiar with the MATLAB® programming environment (The MathWorks, Inc.). The GUI provides a modern, easy-to-use, web-based software interface with the Glenn Research Center's Communication Analysis Suite (GCAS) static link analysis capability. This capability utilizes existing mathematical models to provide analysis and solve for desired outputs. The work outlined in this report describes the development of the interface with a web-based software platform.

Background

The previous interface of the static link analysis component of GCAS implemented the tools in an executable GUI in Python™ programming language (Python Software Foundation). The GUI was designed with Qt Designer (The Qt Company) and written in PyQt5 code, which both have a wide variety of limitations. From a front-end design perspective, the biggest limitations are GUI responsiveness and development. Qt Designer does not support the ability to hide input boxes when they are not needed, limiting the GUI responsiveness. It also lacks support of certain complex input types that require using strings and/or integers, such as calendar and date-time. The use of those types of complex inputs are more desirable for an intuitive GUI design. Qt Designer also makes it difficult to expand the existing GUI because every element is in a fixed location in the GUI layout. To expand the GUI, a developer must modify the layout scheme to add more inputs. In terms of development, the GUI code is cumbersome to expand or change in any way. The GUI and its code does not have support for style elements such as coloring and responsive resizing. The developer must design their layout in the Qt Designer and then compile it into PyQt5 code. The code is organized in an object-oriented methodology that utilizes inheritance to layout GUI elements. This complicates the development of a GUI because simple changes and/or bug fixes become much more difficult to support. Figure 1 illustrates the GUI as previously developed in Qt Designer.

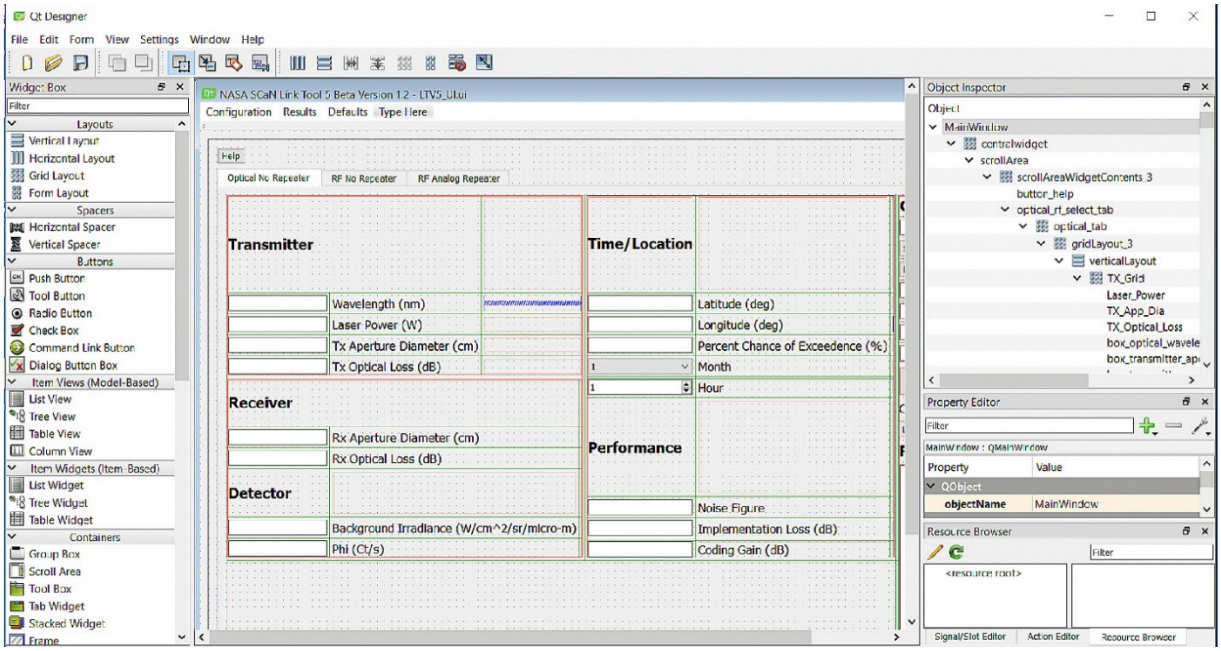

Figure 1.—Static link analysis graphical user interface in Python™ Qt Designer (The Qt Company).

The mathematical models that perform the numerical calculations, originally coded in MATLAB®, were rewritten in Python™ using NumPy and SciPy libraries. This hindered the development times of the previous GUIs and detracted from the interface design. In addition, rewriting the code in Python™ caused a handful of the output values to have limited accuracy (three decimal places) when compared to the MATLAB® code output, treated as the standard output value. This caused concerns about verification testing of the GUI. Since the mathematical model was written in a format that can be packaged as a software component using the MATLAB® Compiler™, it becomes unnecessary to rewrite this code in a different language.

Development Methodology and Discussions

This section describes topics such as code compilation, utilization, web technologies, front-end features, back-end capabilities, deployment solutions and results of the development process.

Code Compilation and Utilization

Because it is a time-consuming process and the possible introduction of errors, preexisting code should not be rewritten if it can be packaged in different programming languages as a reusable software component. A better approach is to package the code for the mathematical models and build the user interface on top of this packaged component. The goal of the GUI is to set the inputs to a function to return an answer. The static link analysis component of GCAS consists of a multitude of scientific details as inputs and outputs, where the output consists of many more elements beyond that of the simple desired result requested by the analysis mode input. Given such aspects of the static link analysis component of GCAS, the MATLAB® code was compiled for use in Python™ using the MATLAB® Compiler™ to act as a single black-box function. Figure 2 illustrates the command sequence to import and initialize those compiled modules in Python™.

The code was integrated into Python™ using the MATLAB® Compiler SDK™, which required installing the compiled files as a Python™ package. This package runs the compiled files on the free MATLAB® Runtime™ without needing a license. Figure 3 illustrates the command to execute the static optical link analysis module of GCAS in Python™.

```
import scanlinktoolV6_Optical_NoRepeater
import matlab
# Runtime engine will terminate when app is closed.
engine = scanlinktoolV6_Optical_NoRepeater.initialize()
```

Figure 2.—Command sequence to import and initialize compiled module in Python™ using MathWorks® MATLAB® Compiler™.

```
output = engine.scanlinktoolV6_Optical_NoRepeater(
    calculation_mode, link_direction, optical_params, los_params, gnd_params)
```

Figure 3.—Command to execute compiled module in Python™ from MathWorks® MATLAB® Compiler SDK™.

TABLE I.—MATLAB® AND PYTHON™ DATA-TYPE CONVERSION

MATLAB® output argument type[a]	Resulting Python™ data type
Numeric array	matlab numeric array object
Double	float
Single	-------
Complex (any numeric type)	Complex
int8	int
uint8	-------
int16	-------
uint16	-------
int32	-------
uint32	int
int64	long
uint64	-------
NaN	float('nan')
Inf	float('inf')
logical	bool
char array (1-by-N and N-by-1)	str
char array (M-by-N)	Not supported
structure	dict
Row or column cell array	list

[a]Scalar unless otherwise noted

This package was integrated into the code as a MATLAB® Runtime™ object with the link tool function as an attribute of that object. The parameters of the function are then structured in Python™ dictionaries, which is the Python™ equivalent of MATLAB® structures. In general, data and data structures are transmitted effectively between Python™ and the compiled MATLAB® files due to the MATLAB® Compiler SDK™ support for Python™ data types and structures, as shown in Table I.

The MATLAB® Compiler SDK™ also supports type parsers for MATLAB® for unique data types such as MATLAB® double matrices, using the "matlab.double([[0.0, –4.0], [180.0, 0.0]])" to represent a two-dimensional double matrix. This wrapper encapsulates nested float lists in Python™, which translates to a 2 × 2 double matrix in MATLAB®. The MATLAB® Compiler SDK™ provides versatility to write applications without needing to rewrite the MATLAB® code.

Finally, the code was tested using test fixtures from the Unittest library (Python™) to ensure that the output results in Python™ were the exact same as MATLAB®. The test fixtures were created using assert equal statements to ensure that the output values in Python™ were exactly the same as expected values brought manually from MATLAB® outputs. The results of this testing were successful.

Web Technologies

Over the last decade, the development of custom GUIs has flagged because of the push for web technologies. In general, a GUI with a predefined layout of input parameters does not provide a design striving for responsiveness, reusability, and expansion. A much simpler approach uses Hypertext Markup Language (HTML) to layout input parameters. HTML also utilizes Cascading Style Sheet (CSS), which gives the developer more control over the style elements of the interface such as coloring and responsive resizing. HTML and CSS are now the standard conventions for designing an interface even if it is not hosted on the web because of the simplicity of development and cross-platform support. HTML is easier to develop and connects into almost any language to manage the logic of the application.

The Python™ Django® web application framework (Django Software Foundation) was used to implement the front-end application instead of using the PyQt5 GUI application framework. Being the oldest web framework, Django® is the world's most popular to develop web applications. The Django® framework strives for simplicity, rapid web development, and minimizing Python™ code. What separates Django® from other web frameworks is that it is a full-stack web framework. Full-stack web frameworks implement front-end and back-end components into a web application for a more dynamic experience. This allows the developer to design the front end and engineer the back end all from the same framework in the same core programming language. This saves time because it eliminates the need to use multiple frameworks to handle different parts of the stack. For example, a dual-framework might use Angular JS for front-end development to manage user interface components and Flask or NodeJS as a back-end framework to manage the server and database components. It was much simpler to use a single framework to handle every part because there was only one person developing the web application.

Front-End Features

The first advantage of using Django® includes support for HTML templates through the Django® template module. Instead of expanding the GUI application's user interface in Qt Designer, a single-page reusable HTML form was designed using the open-source Bootstrap 4 GUI component library. The result was a much richer user interface that was easier to use. The HTML features the latest facets of modern web development conventions and style and uses the most up to date Bootstrap 4 CSS and JavaScript components. The user interface was developed using the Google Chrome web browser and debugged with Chrome's developer tools which give realtime updated errors and warnings which is far more efficient than compiling a user interface.

The second advantage of using Django® includes Django®'s development server that completely mimics a real-time web-server environment. Django® controls each component of an application through views that are written in Python™ and stored on the server. Each view has an associated Uniform Resource Locator (URL) scheme. When the URL opens, the function associated with that view executes on the server. These views handle the responses when the user sends a "GET" or "POST" request. The views also return server responses, render the HTML template, and can send data back and forth between the user interface and the Python™ code. This functionality met the requirements of the interface for the static link analysis components of GCAS.

The third advantage of using Django® includes support for Hypertext Transfer Protocol (HTTP) methods such as GET and POST. The Django® application utilizes both methods in its server architecture. POST is utilized to submit the name and value of each parameter input from the front-end HTML form and send it to the correct server URL upon submission. Django® then receives the data in a view that is associated with the URL where the data was posted. Once the view has the data, it can perform a variety of functions, such as reformatting the inputs to MATLAB® double arrays or floating point numbers, to then support module execution and return the data structure of the link analysis result.

The fourth advantage of using Django® includes the static files module which lets the user bring in CSS and JavaScript files to control interface logic to minimize the code in the Django® views. The JavaScript utilizes existing Bootstrap 4 JQuery files to perform form data validation. The JavaScript also controls interface logic such as showing and hiding configuration parameters when they are not used, saving the form values to the browser's local storage for nonpersistent session storage so they do not disappear after page reload and running extra link tool data validation to ensure the parameter data is completely valid before sending it off in a POST request. Although the JavaScript utilizes local storage to save form values in the current session, local storage is not a means of persistent storage and the values

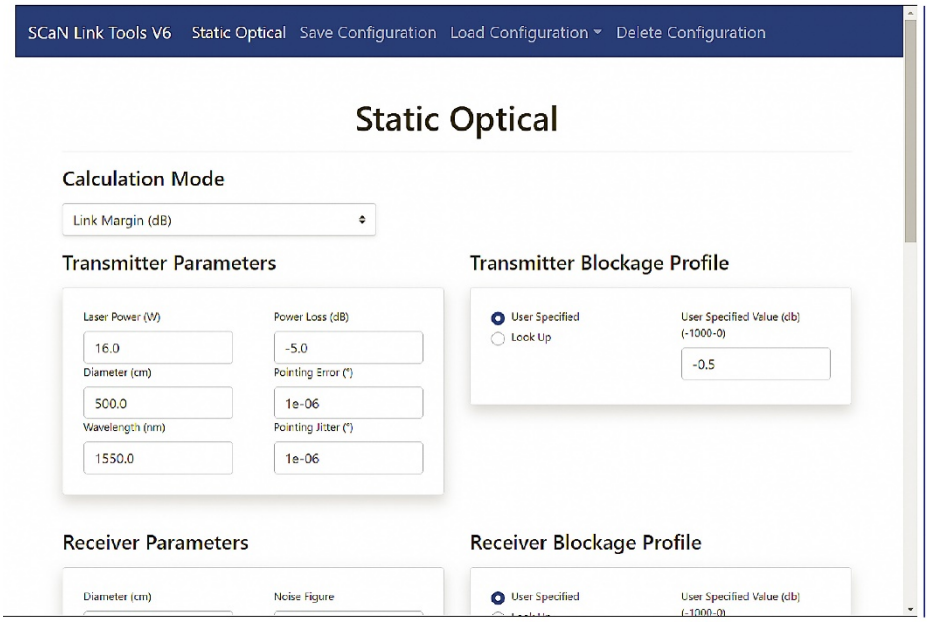

Figure 4.—Front-end interface of static optical link analysis module of Glenn Research Center's Communication Analysis Suite.

Configuration: 'User's Custom Configuration' Saved on: 2019-08-06 21:17:41.799017+00:00

Figure 5.—Server message when user saves configuration.

clear once you clear your browser data. Local storage is a basic key-value database in the browser. Saving multiple custom link configurations for later access was out of the scope of this storage option. Figure 4 illustrates the front-end interface for the static optical link analysis module.

Back-End Capabilities

After completing the front-end capabilities, the application needed a more dynamic means of persistent storage to store the default link configurations and the custom user configurations. This was implemented through Django® back-end capabilities, the fifth and final advantage of using Django®. These back-end capabilities come preloaded in the Django® project. They consist of an object-oriented data model written in Python™ and a lightweight and portable SQLite database that auto builds to match the fields of a particular model.

Save functionality was implemented by reusing the HTML form for the calculations, and instead of executing an analysis, the application saves the parameters to the database upon form submission. The parameters are then stored in the database while the app is closed. Two more parameters were also added to the database: the Custom Configuration Name and the Datetime. These parameters were added so the user can differentiate between the custom configurations when they are loaded from the database at a future date. An example from one of the loaded configurations include "Configuration Name: 'Config5' from: 2019-07-23 21:05:34.388533+00:00." The save function does not save output results because they should be recalculated from the saved parameters to maintain consistency with the current mathematical models of GCAS at all times to make sure that the user is getting the most up to date link analysis results. Figure 5 illustrates the server message for saving a configuration.

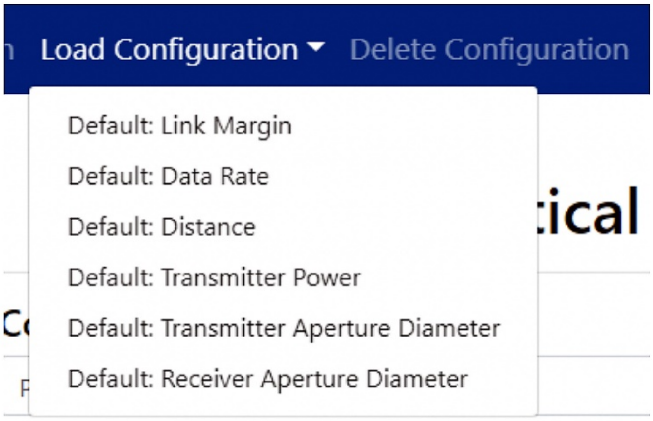

Figure 6.—Load configuration functionality.

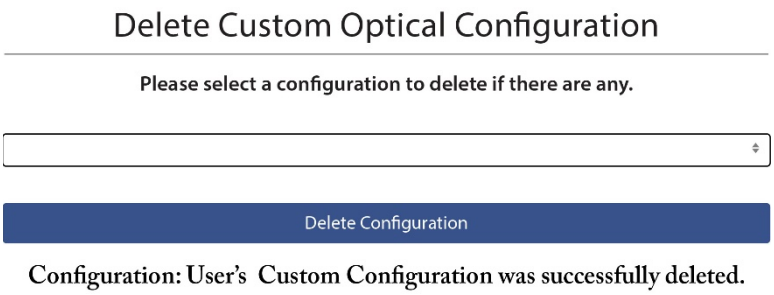

Figure 7.—Delete configuration functionality and server message.

A load function, complementary to the save functionality, was written to pull the configurations out of the database and perform calculations upon user request. The load function was more cumbersome to implement because the data had to be reinjected into the HTML template. The load and save functions both utilize Django®'s QuerySet API, which prevents Structured Query Language (SQL) injection by using Django® methods instead of raw SQL queries. When loading a configuration, the Custom Configuration Name and Datetime appear on the form as a message from the server. This was made possible by Django®'s templating and GET method capabilities. The function, HTTP GET, retrieves data from the database such as the configuration names shown in Figure 6, and renders them on the page with the appropriate URL, to load in the configuration parameter data by configuration name. The database is also preloaded with the default configuration values for static optical and static RF.

Finally, a delete configuration function gives the user the ability to delete added custom configurations. Figure 7 illustrates the delete configuration process and message. Adding back-end functionalities to the GCAS static link analysis web application interface makes it more robust as a full-stack web application.

Deployment Solution

One of the original requirements of the user interface was that the application have a portable executable to run offline. Considering that the standard convention for deploying a web application is hosting it on a production server, it was necessary to replicate the same environment but as a portable executable. The solution to this problem was reengineering an open-source software called Django2exe. Django2exe is a software that turns Django® web applications into progressive web applications that can run as an executable. Django2exe turns on the application in a Chromium Embedded Framework view

that is embedded in a blank PyQt5 GUI pane. Django2exe starts a Chromium web browser in a blank GUI pane that runs the web application. Django2exe was reengineered into Django2Bat, which runs all these processes in a Bat script executable. Django2Bat was stripped of all the original components that came with Django2exe and was upgraded with the portable WinPythonZero 3.5 Python™ interpreter solution able to run Python™, Django®, and the compiled MATLAB® functions. Several modifications were also made to the Django2exe Python™ source code to update all of its components. The Django2Bat solution also uses a Secure Sockets Layer (SSL) encrypted server instead of the original development server so that the web application benefits from Django®'s Hypertext Transfer Protocol Secure (HTTPS) security features. These features include encrypting HTTP GET and POST requests so that the data is not maliciously extracted from the server's port. This deployment solution allowed packaging and running the GCAS static link analysis web application as a progressive web application executable.

Results and Discussions

The web application underwent three major development iterations also known as betas. The first beta was building the front-end static optical link analysis GUI, which was successful but took the longest time because of the methodologies and development of the reusable components described previously. Figure 8 illustrates the results of a static optical link analysis calculation through the entire interface.

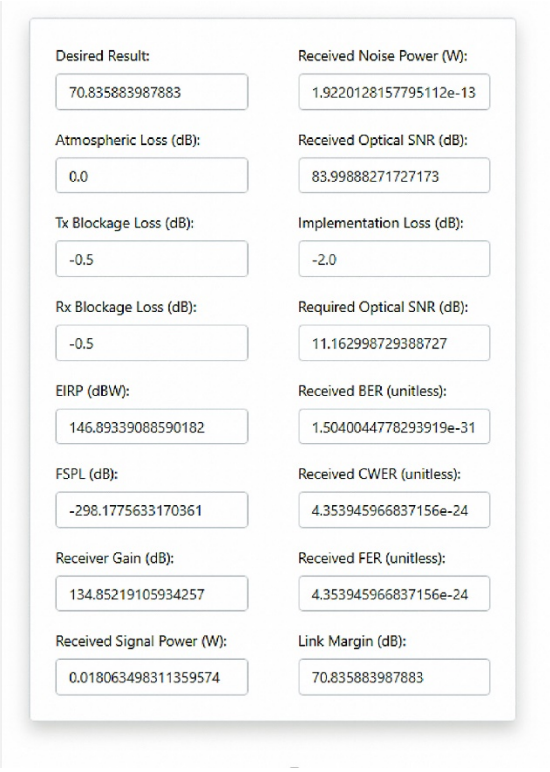

Figure 8.—Static optical link analysis results.

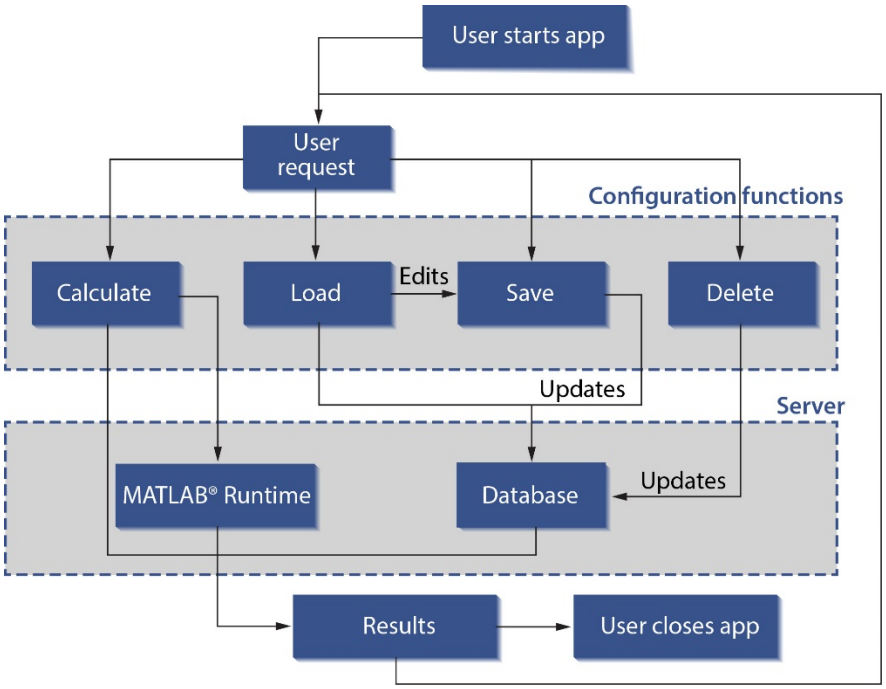

Figure 9.—Use cases of Glenn Research Center's Communication Analysis Suite static link analysis graphical user interface.

The second beta introduced bug fixes and the three back-end capabilities: load, save, and delete to the static optical link GUI. This functionality not only made the web application easier to use but it made it more dynamic as it uses back-end models and databases that can be used for future development such as building a Representational State Transfer Application Programming Interface (REST API). The third major beta introduced the Static RF link analysis GUI and its corresponding back-end model and database with load, save, and delete functionality. This made the GCAS static link analysis GUI more versatile as it now has more use cases for engineers that are not dealing with optical communications.

The third beta was the fastest to complete due to the reusable code from the first two betas. The development time for the third beta was about 5 days compared to the development time for the first and second beta, which was over a month each. This was due to the reusable single-page HTML template developed using Bootstrap 4.0 and about 3,000 lines of JavaScript UI logic that can be reused in most other link tools after static optical and static RF. The faster development time of static RF was also due to the entire Django® application architecture already built for each core function of the web application. All iterations implemented a large number of bug fixes and have been tested against expected values from the MATLAB®-based unit testing routines that output the correct expected values for each link tool. Figure 9 illustrates the possible use cases of the GUI that was developed.

Concluding Remarks

Overall, the implementation of a web-based user interface was successful. From the beginning of the project, the MathWorks® MATLAB®-compiled static link analysis. Glenn Research Center's Communication Analysis Suite (GCAS) functions passed the unit tests in Python™ programming language (Python Software Foundation), which indicated that this implementation would work. Since initial unit testing of the compiled MATLAB® code was successful using Python™ Unittest test fixtures, more link configurations were tested within the application and it maintained higher levels of accuracy

compared to the previous Python™ implementation of the interface. In the context of any space mission or operation, the most valuable asset to using MATLAB®-compiled files is that the results will be exactly the same as executed in MATLAB®. Any user of the GCAS static link analysis front-end application will not have to worry that the data might be different from MATLAB® outputs. This alleviates any problems that might arise in a NASA mission regarding the accuracy of a link analysis computed on the web-based front-end tool.

The efforts of this project will be used to support the upcoming lunar studies to perform analysis for the future communications architecture of the Artemis project. In terms of future development, graphical user interfaces for the GCAS static link repeater tools and/or dynamic link tools can be easily expanded into the web application framework through the previously described methods. In terms of future concepts that can be added into the Django® architecture, a Representational State Transfer Application Programming Interface (REST API) could be built on a more central version of the web application to support additional users of the GCAS capabilities. Users could request link outputs over the internet by sending the parameters in JavaScript Object Notation structures. This would increase the user base of the GCAS capabilities to the widest possible audience.

Bibliography

Bootstrap: 4.3 Documentation. 2019. https://getbootstrap.com/docs/4.3/about/overview/ Accessed March 26, 2020.

Django Software Foundation: Django Documentation. 2019. https://docs.djangoproject.com/en/2.2/ Accessed March 26, 2020.

Django To Exe—Distribute Your Django Project as a Portable Windows EXE Application. Algotronics, 2015. https://algotronics.wordpress.com/2015/10/23/dj2exe/ Accessed March 26, 2020.

Green, Jack L.; Welch, Bryan W.; and Manning, Robert M.: Updating the Space Communications and Navigation (SCaN) Link Tool Executable Software to Version 5. NASA/TM—2019-220234, 2019. http://ntrs.nasa.gov

Herrmann, Michael: PyQt5 Tutorial. fman Build System, 2019. https://build-system.fman.io/pyqt5-tutorial Accessed March 26, 2020.

The MathWorks, Inc.: Handle Data Returned From MATLAB to Python. 2019. https://www.mathworks.com/help/compiler_sdk/python/handle-data-returned-from-matlab-to-python.html Accessed March 26, 2020.

The MathWorks, Inc.: Initialize Package and Return a Handle. 2019. https://www.mathworks.com/help/compiler_sdk/python/mydeployedmodule.initialize.html Accessed March 26, 2020.

The MathWorks, Inc.: MATLAB Compiler SDK. 2019. https://www.mathworks.com/products/matlab-compiler-sdk.html Accessed March 26, 2020.