AFRL-RH-WP-TR-2020-0062

COG-PACK™ REFLECTION BASED USER INTERFACE DEVELOPMENT

Mr. Allen W. Dukes
Airman Biosciences Division

Mr. Courtney J Downs
Mr. Scott J Duberstein
Ball Aerospace & Technologies

May 2020
Interim Report

AIR FORCE RESEARCH LABORATORY
711TH HUMAN PERFORMANCE WING,
AIRMAN SYSTEMS DIRECTORATE,
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by 88th Air Base Wing Public Affairs Office and is available to the general public including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RH-WP-TR-2020-0062 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

FRANCK.DOUGLAS.L.1230132255 Digitally signed by FRANCK.DOUGLAS.L.1230132255 Date: 2021.01.06 16:09:29 -05'00'

DOUGLAS L. FRANCK, DR-III, DAF
Chief, Performance Optimization Branch
Airman Biosciences Division

MCKINLEY.RICHARD.A.1262239710 Digitally signed by MCKINLEY.RICHARD.A.1262239710 Date: 2020.09.03 10:06:43 -04'00'

R. ANDY MCKINLEY
Core Research Area Lead
Performance Optimization Branch
Airman Biosciences Division

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-05-20	Interim	January 2019 to March 2020

4. TITLE AND SUBTITLE
COG-PACK™ Reflection Based User Interface Development

5a. CONTRACT NUMBER
FA8650-16-C-6610

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62202F

6. AUTHOR(S)
Mr. Courtney J. Downs+
Mr.Allen W. Dukes*
Mr. Scott J. Duberstein+

5d. PROJECT NUMBER
5329

5e. TASK NUMBER
08

5f. WORK UNIT NUMBER
H0KG (53290819)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ball Aerospace & Technologies+
2675 Presidential Drive, Fairborn, OH 45324

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Materiel Command*
Air Force Research Laboratory
711 Human Performance Wing
Airman Systems Directorate
Warfighter Interface Division
Applied Neuroscience Branch
Wright-Patterson AFB, OH 45433

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
711HPW/RHBC

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RH-WP-TR-2020-0062

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION A. Approved for public release.

13. SUPPLEMENTARY NOTES 88ABW-2020-2677, cleared 31 August 2020

14. ABSTRACT

In Computer Science, the concept of reflection involves the process of using software to inspect its own implementation, and gather insights into its structure (Sobel & Friedman, 1996). This technique can be leveraged in a number of methods, to help understand the design, constructs, and format of the layout and architecture of software components. Combining the concepts of MVVM (The MVVM Pattern, 2012) and Data-driven decision making (Marr, 2016), may lead to a more efficient implementation of controlling a complex software system. Herein, we describe the use of reflection-based techniques, to streamline and simplify the layout of UI components, based on the internal structure of data-types resident in the Cognitive Operations Gear (COG) Pack architecture (Dukes, et al., 2019).

15. SUBJECT TERMS
COG Pack™, Reflection, UI Development, Software Engineering

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Armando Soto
U	U		SAR	14	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

TABLE OF CONTENTS

LIST OF FIGURES

Reflection Based User Interface Development

Dynamically Generated Configuration-based UI Elements with Example Implementations Found in COG Pack™

Courtney J. Downs, Allen W. Dukes, Scott J. Duberstein

1.0 BACKGROUND

In Computer Science, the concept of reflection involves the process of using software to inspect its own implementation, and gather insights into its structure (Sobel & Friedman, 1996). This technique can be leveraged in a number of methods, to help understand the design, constructs, and format of the layout and architecture of software components. Combining the concepts of MVVM (The MVVM Pattern, 2012) and Data-driven decision making (Marr, 2016), may lead to a more efficient implementation of controlling a complex software system. Herein, we describe the use of reflection-based techniques, to streamline and simplify the layout of UI components, based on the internal structure of data-types resident in the Cognitive Operations Gear (COG) Pack architecture (Dukes, et al., 2019).

2.0 WHAT IS REFLECTION BASED UI, AND HOW MIGHT IT BE USEFUL?

A reflection based User Interface (UI) is one that is capable of creating its own layout and deciding which interface components are required for the task at run time. This is a different approach from traditional UI generation. Traditionally, UI is developed in advance of its use, based on assumed user requirements and workflow. A reflection based UI takes an alternate approach by observing the properties of specific software defined data-types, and generating UI elements in response. This different approach is very important as it may significantly speed up the front-end development of an application by removing the need to program every interface by hand in advance.

3.0 DATA-TYPE MAPPINGS

The first step to creating a reflection based UI is to create a list of rules and concepts that the application leverages to map data-types to interface components. Table 1 describes a starting point of simple rules that could be used for an application to generate its own UI components:

Table 1 - Data-type Mappings

Datatype	User Interface Component
String[]	List<Check Box>
String	Edit Text<String>
String<Object>()	List<Custom React Component>
Integer	Edit Text<Integer>
File Path	Directory Navigation Button
Boolean	Toggle Button/Check Box

COG Pack leverages JSON as a simple data-exchange format to implement this rule chart. Figure 1 is an example of JSON data and the interface components that can result.

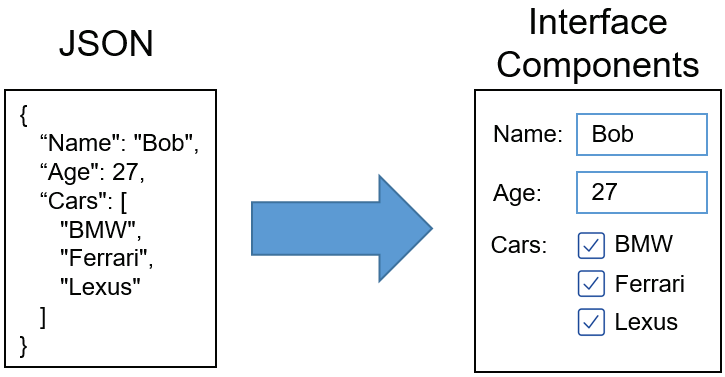

Figure 1 – Simple Interface Elements Derived from JSON

The JSON data provide to the application will need to either contain **Sample Data** (as shown in Figure 1) or have **Labelled Data Types.** None of the fields in the JSON are allowed to have **null values**. These requirements allow the application to assign interface components based on datatypes in a very straightforward way using the JavaScript method called "**typeof**" along with the programmed rules listed above in Table 1.

Additional JavaScript reflection based methods useful in parsing the JSON data and generating interface cards are shown in Table 2:

Table 2 - Additional Javascript Reflection Methods

Method	Use case	Return value
Array.isArray()	Array.isArray({Cars: ["BMW", "Ferrari", "Lexus"]})	true
Object.keys()	Object.keys(JSONFromFigure1)	["Name", "Age", "Cars"]

4.0 IMPLEMENTATION

The COG Pack Dashboard currently leverages the React.js framework (Gackenheimer, 2015). React.js defines all reusable UI elements as components. Derivatively, COG Pack describes the dynamic cards created through reflection as a **Dynamic Sensor Component**. This component interprets the data from a given configuration file or string (Dukes, Duberstein, & Rommel, Cognitive Operations Gear (COG) Pack(TM) API Specification, 2020) and populates itself with related interface elements using the mappings as defined in Table 1. The main classes involved in this process are listed and described in Table 2.

Table 3 - Dynamic Sensor Component Classes

Class Name	Class Purpose
App.js	Parent class containing the Tab Menu component
TabMenu.js	Child class that contains the Dynamic Sensor Components
DynamicSensorComponent.js	The React component that can automatically populate its UI elements.

The diagram in **Figure 2** explains the flow of JSON data through the user interface and each class's responsibilities that generate the interface components.

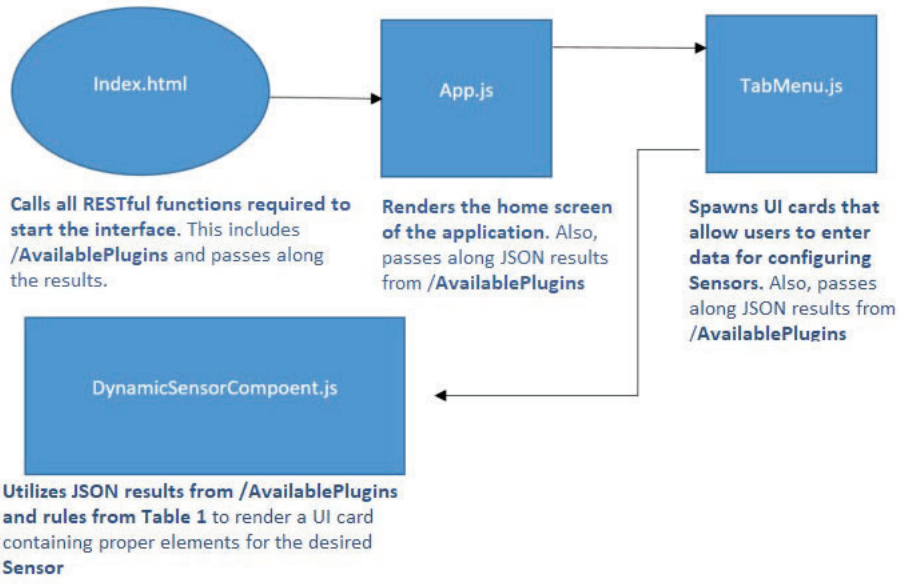

Figure 2 - User Interface Data Flow

4.1 Leveraging the RESTful COG Pack Implementation

A description of the data-types provides the required information to populate the Dynamic Sensor Component UI elements. Additionally, a process to query this information requires implementation. The COG Pack Dashboard leverages the RESTful implementation functionality found in COG Pack (Duberstein, Dukes, Blackford, Downs, & Rommel, 2020) as outlined in the following sections.

4.2 Available Plugins

Executing the ***AvailablePlugins*** REST/GET command, http://localhost:8088/AvailablePlugins, returns a data packet composed of a JSON string value. This value contains a list of Sensors with their respective properties. Figure 3 displays an example of this data.

Figure 3 - JSON Sensor and Signal(s) Object Data

4.3 Dynamically Populating Available Sensors

Upon instantiation, the COG Pack Dashboard first executes the "**ReactDOM.render**" function inside of **index.html** which then performs the call to ***/AvailablePlugins.*** **App.js** receives these results via react component props, then passes the results to TabMenu.js class also via component props. **TabMenu.js** uses this data to populate the list of available Sensors that a user can assign to a participant, as shown in Figure 4. TabMenu.js eventually passes the results from ***/AvailablePlugins*** to the **Dynamic Sensor Component**.

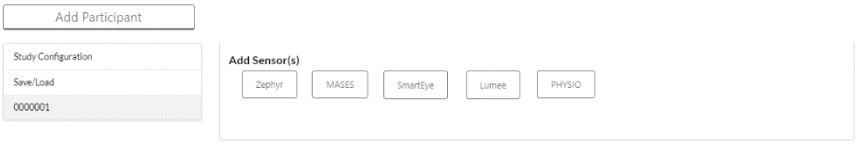

Figure 4 - Dynamically Populated Sensor Buttons

4.4 Dynamically Populating Individual Sensors

The configurable properties of each Sensor in COG Pack is described in the JSON data returned by the ***AvailablePlugins*** call. TabMenu.js references that JSON data to create the Dynamic Sensor Card for each Sensor, resulting in the creation of a **Dynamic Sensor Component**.

4.5 Dynamic Sensor Component

The **Dynamic Sensor Component** is the COG Pack implementation of the React.js component that manages the interface element generation. This Component will use the logic from Table 1 to automatically create UI elements based on the datatypes from the data that is passed into its parameters. A function inside of TabMenu.js will create the Dynamic Sensor Card objects and pass them the required parameters.

The most important parameter that will be passed into this component will be the sensor configuration JSON string for the given sensor type (example JSON data shown above in **Figure 3**). That value will be passed as the "parent State" property. It will be used to keep track of changes that are made through the interface for that specific sensor.

4.6 Adding Card Functionality

Having explained the process of generating UI elements, adding functionality is the next step.

After creating any UI component of a specific type, a function is assigned that specifies what happens when manipulating the component (clicked, checked, text value adjusted, etc.) React accomplishes this by assigning functionality to the components property for onClicked() or onChange(). Passing an appropriate function to either of these parameters will need to adjust the component's "state" (list of local variables) with the same title as the adjusted value.

Example:
After interpreting the Sensor's JSON configuration string and determining which of the fields match the conditions in Table 1 for a CheckBox, the proper **Dynamic Sensor Component** would trigger a function similar to the one in Figure 5**.** The purpose of this function is to add a check box to the proper interface component.

```
/**
 *  fiedName - Sensor configuration property name.
 *          (i.e. Signals)
 */
this.addCheckBoxElement = (fieldName) => {

    let cboxes = [...this.state.checkBoxSection];
    cboxes.push(
        <Checkbox
            label={article}
            key={index}
            style={checkBoxStyle}
            defaultChecked={false}
            onClick={(event) => this.updateCheckBoxes(event, fieldName)} />)

}
```

Figure 5 - Add CheckBox Element Function

Each value type (Check Box, Edit Text, Button, etc.) will have a corresponding function. These functions will update the state variable of the current component using the first argument as the property identifier:

Ex. onChange(updateString(“Sensor ID”))

Ex. onClick(updateStringList(“Signals”))

The creation of a 'Check Box' component for a property requires implementing its onChange() event handler. This event handler executes when that specific check box is accessed. Figure 6.

```
this.updateCheckBoxes = (event, fieldName) => {

    let tempCopy = [...this.state.configurationFile[fieldName]];

    // Flip the checked/enabled status.
    let index = tempCopy.indexOf(event.target.value)
    tempCopy[index].Enabled = !tempCopy[index].Enabled;
    this.state.configurationFile[fieldName] = tempCopy;
}
```

Figure 6 - CheckBox onChange Event Handler

The key element that allows this function to be dynamic is the ability to use the variable 'fieldName' as the key in the state variable object. This functionality is handy because the prior function retrieves the value is only required to assign the value of the literal name belonging to the JSON configuration item from the given Sensor.

4.7 Resultant UI

The end result of the previous steps is the creation of a UI based primarily on the data-types that require editing and manipulation. This helps a developer manage the elements of a plug-in based architecture. This formulation creation of a UI by a developer can also benefit the user, through facilitating a common look and feel for independent software packages that may share common attributes. This common look and feel can be seen in both the UI User's manual (Downs & Dukes, 2020) and the Sensor Setup video (Downs, 2020).

5.0 CONCLUSION

The design concepts and processes found in this style of reflection-based UI generation serve as an example of creating dynamic user interfaces and expand upon the MVVM pattern. While still following the concepts of separating the UI controls from the business logic, reflection-based UI creation may be an additional step into providing a unique UI design concept. Referencing the underlying data-types allows the application of data-driven design concepts, and may provide an overall faster response to evolving requirements than the typical reliance on hand-jammed knee-jerk UI development.

6.0 REFERENCES

Downs, C. J. (Writer), & Downs, C. J. (Director). (2020). *Sensor Setup* [Motion Picture]. 711 HPW/RHBCN.

Downs, C. J., & Dukes, A. W. (2020). *COG Pack™ Dashboard.* Wright Patterson: DTIC.

Duberstein, S. J., Dukes, A. W., Blackford, E. B., Downs, C. J., & Rommel, M. T. (2020). *COG Pack™ Technical Design Document.* RHBCN, 711 HPW. Wright Patterson: DTIC.

Dukes, A. W., Duberstein, S. J., & Rommel, M. T. (2020). *Cognitive Operations Gear (COG) Pack(TM) API Specification.* RHBCN, 711 HPW. Wright Patterson, OH: DTIC.

Dukes, A. W., Duberstein, S. J., Blackford, E. B., Klosterman, S. L., Rommel, M. T., & Downs, C. J. (2019). *Cognitive Operations Gear (COG) Pack™ Software API White Paper - An API Designed to Enable the Integration of Multi-modal Data-types for Real-time Analysis.* RHBCN, 711 HPW. Wright Patterson, OH: DTIC.

Gackenheimer, C. (2015). Introducing Flux: An Application Architecture for React. In C. Gackenheimer, *Introduction to React* (pp. 87-106). Berkley, California: Apress.

Marr, B. (2016, 06 14). *Data-Driven Decision Making: 10 Simple Steps For Any Business*. Retrieved 2020, from Forbes.com: https://www.forbes.com/sites/bernardmarr/2016/06/14/data-driven-decision-making-10-simple-steps-for-any-business/#3a621be95e1e

Miller, K. (2019, 08 22). *Data-drive Decision Making: A Primer for Beginners*. Retrieved from Northeastern University: https://www.northeastern.edu/graduate/blog/data-driven-decision-making

Rommel, M. T., Dukes, A. W., & Blackford, E. B. (2020). *COG Pack™ Eye/Head/Gaze/Heatmap 3D Display Documation.* Wright Patterson: DTIC.

Sobel, J. M., & Friedman, D. P. (1996). *An Introduction to Reflection-Oriented Programming*. Retrieved from Citeseerx: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.8581

The MVVM Pattern. (2012, 10 04). Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

COG Cognitive Operations Gear

UI User Interface