Abstract:
The AFIT Directed Energy and Atmospheric Models (ADAM) are utilized by military and DoD research departments for the modeling of high-energy laser systems and engagements. The current models are effective but lack a modern design strategy. This makes code modifications difficult and causes the models to operate inefficiently at run time. To remedy this, ADAM will be restructured according to guidelines set by Gamma et al. in Design Patterns: Elements of Reusable Object-Oriented Software. In addition, the user will have the option to run ADAM programs using C++ as opposed to MATLAB.

Improvement of ADAM’s Behavioral Design Pattern & Conversion of Modeling Capabilities from Matlab to C++
Brannon Elmore1, Dr. Steven T. Fiorino1
1Department of Engineering Physics, 2Southwestern Ohio Council for Higher Education (SOCHE), 3The Ohio State University

Motivation:
ADAM users have specific requirements regarding what information they want from the software and how many times they want to use its capabilities. The current model, ADAM V4, does not have the flexibility to adapt to the needs of the user. The next version of ADAM will have the ability to run in either MATLAB or C++, only calculate what the user requests, and it will bypass recalculating data if the user has not specified a change to inputs that either directly or indirectly affect the results. This both reduces memory costs and its runtime by removing the need to copy memory for storage while recalculations are made and bypassing function calls which have already been utilized.

Methodology:
The strategy pattern used in the new version of ADAM employs a more flexible use of algorithms by writing them for general use and encapsulating them as opposed to inheriting objects or classes in a program. Through isolating the algorithms, unintentional dependencies on other objects or user-input can be avoided. This results in a code that is independent of its context and can easily be modified without causing unwanted changes to the software as a whole. To reduce the run-time of the program, the option of running models in C++ will also be added to the newest version of ADAM. Because MATLAB is a higher-level language, its run time is significantly impeded as the size of a program grows (see Figure 2 below). C++ is a lower-level language that requires less refinement when translated to machine code, and it lacks the JIT compiler that MATLAB can cause. C++ conversions will continue to be made to prevent the run-time slowdowns experienced in MATLAB.

Conclusion:
Through the implementation of a modern programming strategy, the dependency issues in ADAM V4 will be alleviated. To further optimize the latest version of ADAM, users will also have the option to run base-level functions in C++.

Looking ahead:
The new ADAM updates are still in the early phases of development. In future months, the next stages will include further refinement of the strategy pattern to calculate all possible values using expertData.m. The disadvantages this presented were a tradeoff between flexibility, wasted memory, and slow run time.

References:
• Gamma, E., Helm, R., Johnson, R., and Vlissides, J.
• Eric Freeman, Elizabeth Freeman, Kathy Sierra and Bert Bates, Head First Design Patterns, First Edition, Chapter 1, Page 24, O'Reilly Media, Inc, 2004.

Analysis:

Contact information:
Kaitlin L. Keltos
klkeltos@gmail.com
(937) 581-6440

Figure 1: Strategy Pattern Analysis – The method of transportation is chosen by the user and independent of the fact that the user wants to travel to the airport.

Figure 2 (left): MapStandardLutToType.cpp – A portion of the 187 lines of code in the current ADAM software that calculates all possible values using expertData.m. The disadvantages this presented were a tradeoff between flexibility, wasted memory, and slow run time.

Figure 3 (right): ADMs.m – A portion of the 187 lines of code in the current ADAM software that calculates all possible values using expertData.m. The disadvantages this presented were a tradeoff between flexibility, wasted memory, and slow run time.

Figure 4 (left): MethodSignature.pdf – The finalized code in Figure 4 was converted to C++ through Visual Studio. With the help of the Armadillo library, many conversions to C++ are straightforward.

Figure 5 (right): MapStandardLutToType.m – The updated code, shown utilizing the vito broken into 138 functions that make individual calculations with the expertData information, allowing users the flexibility to select exactly which features they wish to calculate.

Figure 6 (left): Run-Time Comparisons – Graphical data comparing the computational speeds of the same program in different languages. C++ had the best run time, while MATLAB came in 3rd or 4th by a small margin depending on the size of the code.