3.4 PROCESS AND TECHNOLOGIES FOR THE TRANSITION OF RESEARCH ALGORITHMS TO OPERATIONS FOR REAL-TIME SATELLITE PROCESSING

Alex Werbos*, Erik Steinfelt, Jordan Bentley, Edward Kennelly, David Hogan, Hilary Snell and T Scott Zaccheo
Atmospheric and Environmental Research, Inc., Lexington, Massachusetts

1. BACKGROUND

The effective transition of the latest scientific research to operations in the weather and remote sensing community has long been recognized as a challenge (Serafin, 2002). In the traditional process for operational ground processing system development, a custom infrastructure is developed for each new mission or sensor. In order to operate within this new infrastructure, scientific algorithm software will need to be engineered to meet the required interfaces. As part of this process, two design factors will drive the larger goals of minimizing development cost: the capability for updates to operationalized algorithm software to be easily and accurately tested in a development environment, and the ability of that software to fully-utilize the processing resources of the operational environment.

Over the timeline of system development, deployment, and operations, it is to be expected that updates to the algorithm software, driven both by science and engineering concerns, will be required. If these updates must be staged and tested within a copy of the operational environment, the time and resources required for each update will be substantially increased. If a lightweight development environment is used, individual domain experts and software engineers will be able to quickly test algorithm updates and verify the correctness of their outputs. In order to ensure the integrity of these tests, the development environment must be capable of using the latest version of the operational software. Otherwise, the interactions between the operational differences and the development updates may generate errors not observed in the development environment. Therefore, we must evaluate the operationalization process with an eye toward "round trip" algorithm code migration between the final deployed infrastructure and individual workstation tests.

Every operational system is subject to certain constraints on processing latency and available computational resources. In order to meet these demands, the scientific algorithms must effectively-utilize infrastructure resources. This requires algorithms to effectively scale to the available number of processing cores, and avoid unnecessary inefficiencies in I/O. In modern cluster-processing systems, this will typically require algorithms to not only be capable of parallel processing, but to split that processing across multiple computational nodes. To keep network load manageable, the algorithms

There are two approaches that are typically taken to transition the algorithms from research to operations: algorithm software can either be adapted from existing sources, or be implemented from scratch. While either of these approaches can produce effective ground systems, both can introduce significant overheads and risks.

When taking the approach to re-engineer existing software for the new platform, developers may choose to either directly modify the existing code to meet the necessary interfaces, or develop an adapter that links the operational infrastructure to the science algorithm. These approaches have the advantage of relatively low up-front cost; leveraging existing code for algorithm processing reduces the amount of development. However, maintaining this low investment will require leaving the core algorithm logic unchanged. This may result in an algorithm that is not optimized for the operational environment, increasing the risk of latency issues and additional hardware cost. In

* *Corresponding author address:* Alexander Werbos, Atmospheric and Environmental Research, Inc., Lexington, MA; email: awerbos@aer.com

the case where an algorithm wrapper is written, this problem may be compounded by forcing multiple translations of data objects on read and write to shift between infrastructure and algorithm formats. If the algorithm code is updated, this problem may be mitigated, but a separate software baseline must be created for the operational code, breaking the link with the existing scientific algorithm. This may make it difficult to integrate changes from developers working in a different environment.

In the case where algorithm software is rewritten from scratch, a substantial up-front investment is required, but the resulting software can be written to be effectively executed within the new infrastructure. Nevertheless, there is still the need for a separate operational software baseline that may differ from that used in scientific development. If the software becomes bound too tightly to the operational infrastructure, it may be very difficult for the new codebase to be effectively updated by users working in a different environment, which will limit the avenues by which these updates can be made, and increase their cost.

When we consider these three typical approaches to transitioning research algorithms to operations (Figure 1), we see that while each of them are capable of producing an effective system, they also involve tradeoffs. Up-front development, maintainability, and hardware requirements must be kept in careful balance to ensure project success, and stakeholders must be aware of the way the compromises will manifest in the resulting operational platform.

2. SYSTEM ARCHITECTURE

In this work, we present a system architecture where specially-designed scientific algorithms are developed as encapsulated components that can be easily be operated both in different software environments and well as across multiple missions with different sensor and performance characteristics. This approach reduces the risk and overhead that are present during the typical ground processing system development process.

We propose a system where algorithms exist as a single software package, written to interfaces that are implemented for execution both within test and operational environments. Each of these algorithms is designed to isolate and abstract platform-specific factors, such as sensor characteristics and process block size. These algorithms are augmented with programmatically-accessible information describing their inputs and outputs, allowing general-purpose software to analyze, manipulate, and display algorithms and data without additional manual intervention.

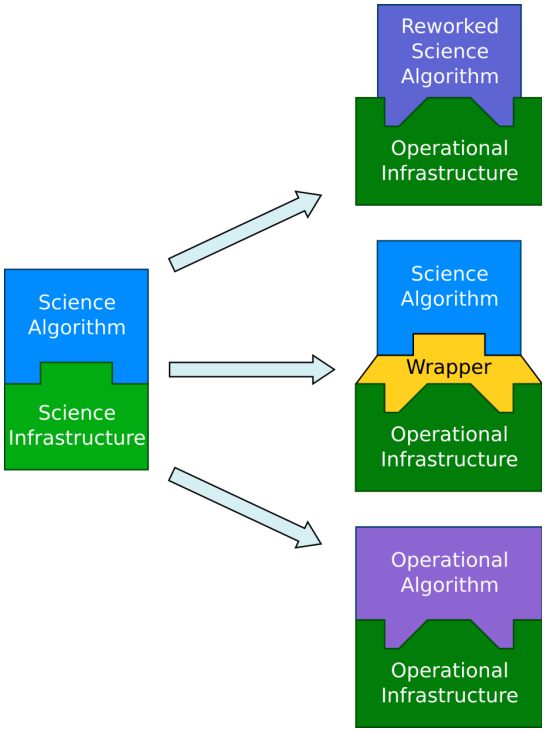

Figure 1. Three traditional research to operations transition paradigms

By ensuring that algorithm code is implemented to use general-purpose interfaces, and that the associated algorithm science can be driven by inputs from a variety of platforms, we are able to put forward the notion of an algorithm library. Algorithms, once developed to these frameworks, can be easily migrated not just between different environments within a single project, but across multiple missions (Figure 2). These algorithms can be viewed as selectable components, where the process of building a ground system is principally one of assembling and configuring the necessary off-the-shelf algorithm code. In so doing, new missions are able to leverage the considerable investment of prior development, not just in initial investment, but in years of cumulative bug-fixes and improvements as well.

Using the programmatically-accessible metadata concerning algorithms and their outputs, this concept can be further extended to supporting tools and applications. It is possible to write tools that ingest the documentation provided about algorithm data and their formats, and then

automatically translate them to the desired final outputs. Document generation tools can analyze the algorithm system configuration and produce standardized materials required for the development and delivery process. Together with the standardized algorithm components, these reusable tools allow users to benefit from the maturity and stability of software that has been tested and improved through multiple software lifecycles, reducing time and cost both for initial system implementation, as well as throughout the maintenance and operations cycle. This concept is illustrated in Figure 3.

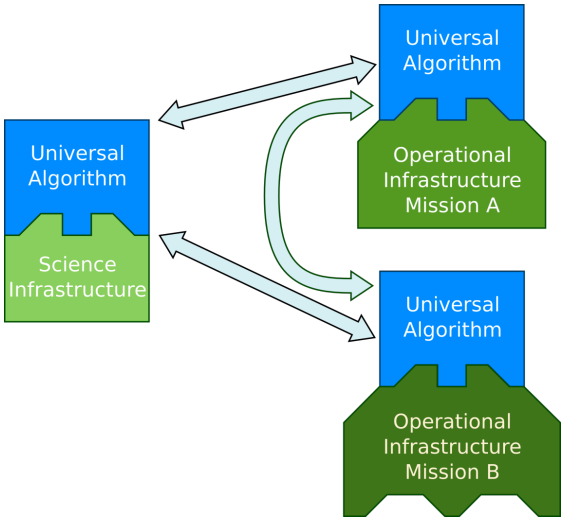

Figure 2. Software architecture concept for multi-mission algorithm support

3. SOFTWARE IMPLEMENTATION

Each of the concepts that we have introduced in this system have been implemented, in some form, over the course of multiple projects. Here we discuss how the general principles of algorithm encapsulation and programmatically-accessible data traits have been manifested in concrete software packages.

In our architecture, algorithm encapsulation is provided by the use of well-defined interfaces for all algorithm interactions with the processing infrastructure. The two interfaces by which this is accomplished are the Algorithm Execution Interface, and the Data Model Interface (DMI). In the Algorithm Execution Interface, each algorithm is developed to operate on a generalized execution area known as a "context". A context contains a unique specification of the data the algorithm should generate. For most image processing algorithms, this will be a timestamp identifying the image to be processed, and a range of pixels that define the block within that image. By making this execution area a parameter given to the algorithm at run-time, the infrastructure is free to divide algorithm executions across multiple blocks and processing hosts. This balancing can even be dynamic, in cases where processing load is anticipated to fluctuate.

The DMI is the software interface used for all algorithm input and output from the mission's infrastructure. As part of its execution, a scientific algorithm will read all parameters and inputs from the DMI. When the algorithm has completed its execution, all outputs will similarly be passed to the DMI for persistence in the infrastructure. The exact datasets and areas queried and written will depend on the context passed to the algorithm. As long as the underlying infrastructure provides an implementation of the DMI, the algorithms can be moved from one environment to another without requiring any changes to their input and output code. Implementations of the DMI have been developed both for file-based workstation testing operations, as well as high-performance clustered systems designed for operational processing, and algorithms have been

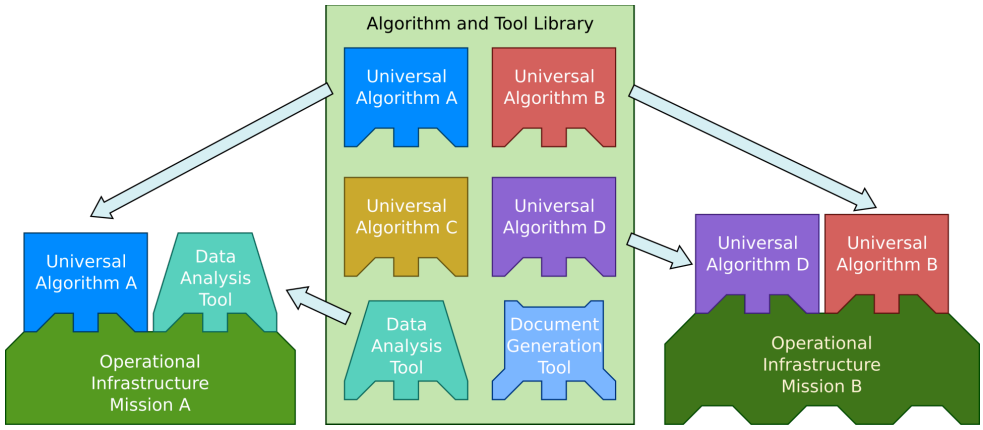

Figure 3. Software architecture concept for library of reusable algorithms and tools

successfully transitioned back and forth between them.

In order to ensure the capability for seamless algorithm transitions, most algorithm implementations are designed to be stateless, and write out all necessary temporary data to the DMI. This allows the underlying infrastructure maximum freedom in initializing and destroying algorithm instances. Instead of needing to keep track of which algorithm is assigned to a particular processing node, algorithm processing can be freely and dynamically divided between nodes, without loss of data.

To provide the capability for programmatic access to algorithm and data characteristics, we have developed the Algorithm Data Descriptor Database (ADDB). This system consists of XML description files associated with each software component that describe the associated algorithms and data types. These files are processed by the ADDB library into a database of metadata accessible to client programs through an API. Each XML file associated with a particular component can be loaded as an individual fragment; there is no requirement for a centralized manifest file. Instead, any references to external data types or objects will remain unresolved until a fragment containing those data types is read. This allows a great deal of flexibility in distribution, as a fully-functioning ADDB can be fully constructed from any given subset of components; users are not required to import and operate with the complete library of available algorithms and tools.

In addition to the descriptions of the immutable properties of the algorithm code and associated data, the ADDB also provides an API for creating system configurations from those algorithms and types. These system configurations describe how the available algorithms and tools are assembled into a particular data processing system. Tools developed using the API provided by the ADDB can ingest user-required outputs and automatically generate a system configuration that generates those outputs using the algorithms available in the ADDB. Other tools, built to read those system configuration ADDB files will be able to automatically configure themselves to display and analyze the outputs of the system. Using both layers of ADDB system, users will be able to easily assemble effective processing systems, and explore the operation of those systems.

4. SCIENCE ARCHITECTURE

Figure 4 illustrates a typical evolutionary process where science algorithms evolve in parallel but separately from the science basis/ research algorithms. This process leads to a number of problems. First the transition of a science algorithm to the operational environment is usually a one way process. The ability to incorporate new science capabilities in hampered because of the disparate software environments and software baselines between the research and operational environments. Furthermore as new systems are developed, algorithm and associated research and operational software are typically adapted in ways unique to that specific system and software infrastructure. The result is increasing fragmentation in the science and software baseline.

Often there is a high degree of commonality in the science basis for the various systems. An approach that abstracts these common features and develops configurable algorithm components works hand and glove with the software architecture described in Figure 3. Such an architectural approach, common in the software world, is less familiar to science algorithm developers. However it has the potential to

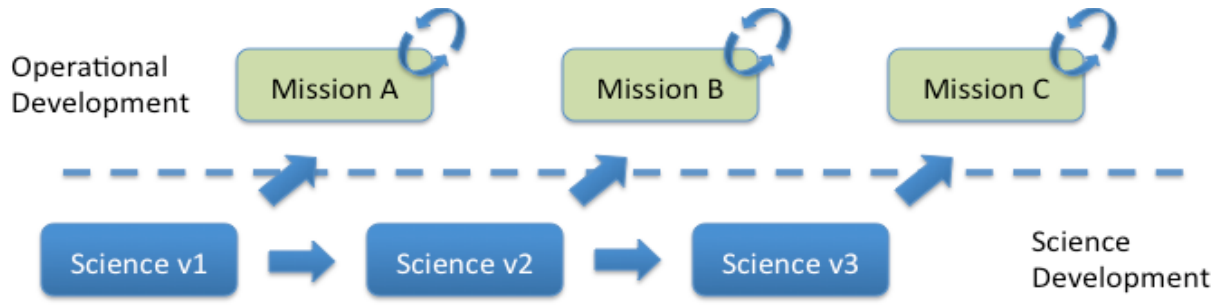

Figure 4. Evolution of operational ground processing systems

significantly improve the overall productivity of the process.

5. CASE STUDIES

The architecture we present had its core developed as part of the GOES-R program, and has been developed internally since then. This section describes how these efforts have implemented the architecture we describe, and the effectiveness of the resulting systems.

5.1 GOES-R

In the GOES-R project, the scientific algorithms were written from scratch for the new system based on Algorithm Theoretical Basis Documents (ATBDs) that described the algorithm functionality. Early on in the process, the decision was made to engineer a technical framework that would permit the scientific algorithms to be executed on meaningful datasets, without change, in both the cluster-based operational environment, and the development workstations used by algorithm developers. It was here that the DMI approach and the notion of algorithm execution contexts were first adopted. In this case, a lightweight infrastructure, based on Python and HDF5, was implemented to drive algorithms using data of the same size and type as would be processed in the final system.

GOES-R also implemented a structured process for migrating the science algorithms as documented in the ATBDs to operational software featuring tight integration of the science and software disciplines throughout the lifecycle (Kennelly, 2013). This process was coordinated through the creation of an Algorithm Baseline, a set of documents that centrally managed all of the information regarding algorithms, their performance, and the data types they used. This function of the Algorithm Baseline is analogous to a manually-managed version of the ADDB. By maintaining and referencing these materials, the process ensured that the science integrity of the algorithms was retained at all steps in the development. It employed joint peer reviews (science and software teams) during the design phase and thorough testing through the code and unit test (CUT) and integration and test (I&T) phases with strict acceptance criteria. The testing encompassed both the science integrity including product accuracy and software quality/ performance.

The validity of these approaches has been borne out by the success of the GOES-R development process (Kalluri, 2014). Much of this success can be attributed to the ease with which distributed, specialized teams have been able to address their issues within a minimal environment, while relying on the interface covenants with other components to ensure that the tested updates will perform correctly in all situations.

6. MULTI-MISSION ALGORITHMS

We have recently been working to extend GOES-R algorithms, originally designed to operate in geosynchronous orbits with the Advanced Baseline Imager instrument to also function in a Highly Elliptical Orbit being considered for upcoming missions (Garand, 2009). We developed the multi-mission (interoperable) algorithm as configurable components that fit within the overall software architecture described in this work. These results were reported separately (Hogan, 2014). We recognize that there will be cases were system/ instrument differences will be too great and that separate developments paths may be required. We anticipate that this will occur less often than may be thought and that a discipline that seeks to exploit commonality in the development of configurable algorithm components will reap many rewards. The multi-mission approach is illustrated in Figure 5.

7. SUMMARY

We have described an architectural approach to

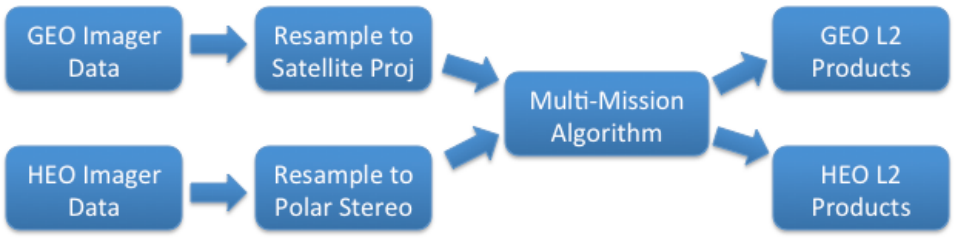

Figure 5. Operational concept for multi-mission algorithm use

the software and science development of remote sensing algorithms. Key features include a common data model interface across development, test and production environments, configurable algorithm components and a multi-mission approach to the science algorithm development. This approach has the potential to improve the science to operations process with the result of transitioning the latest science capabilities faster and with less expense than currently.

8. ACKNOWLEDGEMENTS

AER wishes to acknowledge the support of the Harris Corporation for their support of the multi-mission algorithm development.

9. REFERENCES

Hogan, D., A. Werbos, J. Bentley, E. Kennelly, E. Steinfelt, T. S. Zaccheo, and W. Davis, 2014: Multi-Mission Remote Sensing Ground Processing Algorithms, *Tenth Annual Symposium on New Generation Operational Environmental Satellite Systems (J1.4), 2014 American Meteorological Society Annual Meeting*, Atlanta, GA

Garand, L. and G. Kroupnik, 2009: The Polar Communications and Weather (PCW) mission: a Canadian project to observe the Arctic region from a highly elliptical orbit. Presentation. *American Meteorological Society Annual Meeting, Phoenix*, AZ, Amer. Meteor. Soc., 16SATMET 1.2, https://ams.confex.com/ams/89annual/techprogram/paper_145382.htm

Kalluri, S., R. Kaiser and D Vititoe, 2014: Lessons Learned From Implementing Operational Algorithms for Product Generation During GOES-R Ground Segment Development, *Research to Operations Pathway for Satellite Data Retrieval Algorithms (J2.1), 2014 American Meteorological Society Annual Meeting*, Atlanta, GA

Kennelly, E., T. S. Zaccheo, C. Botos, E. Steinfelt, H. E. Snell, and R. Khanna, 2012: Reproducibility of Research Algorithms in GOES-R Operational Software. *Ninth Annual Symposium on Future Operational Environmental Satellite Systems*, San Francisco, CA

Schmit, Timothy J., Mathew M. Gunshor, W. Paul Menzel, James J. Gurka, Jun Li, A. Scott Bachmeier, 2005: Introducing the Next-Generation Advanced Baseline Imager on GOES-R. *Bull. Amer. Meteor. Soc.*, **86**, 1079–1096.

Serafin, Robert J., Alexander E. Macdonald, Robert L. Gall, 2002: Transition of Weather Research to Operations: Opportunities and Challenges. *Bull. Amer. Meteor. Soc.*, **83**, 377–392.