J10.4

AUGMENTING SADT WITH UML

A HYBRID APPROACH FOR THE DESIGN OF OPERATIONAL SCIENCE ALGORITHMS

John L. Baldwin *, Alexander Werbos, and T. Scott Zaccheo
Atmospheric and Environmental Research (AER), Inc.

1. INTRODUCTION

Science algorithms perform calculations for modeling physical phenomena and estimating physical properties. They take input and control data to generate output data. Implemented in software, science algorithms are functional in nature and generally not user interactive. As such they can be designed as modular computational units which may be run independently, as part of batch processing, or as components within a user or service interactive environment. Two contemporary approaches to address the analysis and design of such systems are Structured Analysis/Structured Design and Object Oriented Analysis and Design. Structured Analysis and Design Technique (SADT) is a specific methodology which emerged out of the variety of different structured analysis and structured design approaches. SADT was designed to simplify and provide a consistent model to describe hierarchical systems. Unified Modeling Language (UML) is a specific object oriented design methodology, serving a similar role to object oriented systems which SADT provides for hierarchical systems. Collaborative computing and ever larger and more complex operational scientific software systems create greater demands on software and systems engineers to reduce risk and provide greater features while working to keep costs as low as possible. Augmenting UML with SADT provides more in-depth algorithm analysis and functional decomposition, while still retaining the strong object model that facilitates large-scale system design.

Large scale software engineering projects are complex, expensive, and inherently risky. Clear communication is critical to meet requirements, stay in budget, and minimize risk. Effective design is critical to clear communication. SADT provides a common language for scientists and software engineers to communicate ideas on how the algorithms function. First and foremost, SADT provides a common language to demonstrate algorithm correctness.

* *Corresponding author address:* John L Baldwin, AER, Inc., 131 Hartwell Avenue, Lexington, MA 02421-3126; email: jbaldwin@aer.com

This paper proposes a hybrid structured analysis and object oriented design approach using SADT with UML toward developing science algorithms and documenting legacy algorithms for incorporation into operational systems and environments. We present an introduction to GOES-R and its systems and software engineering for context. This is followed by an introduction to SADT along with its strengths and drawbacks, a brief introduction to UML, a working example of SADT, and then demonstrate how UML augments SADT for a robust algorithm development approach.

2. GOES-R OVERVIEW

The National Oceanic and Atmospheric Administration's Geostationary Operational Environmental Satellite system is a premier national asset with regard to weather and climate analysis. GOES satellites stay in a fixed orbit above the Earth, providing a constant stream of data. These data are applied in areas such as monitoring for a variety of severe weather conditions, hazardous conditions such as volcanic ash for aviation, fire and smoke analysis, and many areas of climatology research.

GOES-R is the next generation of the GOES Mission. GOES-R is comprised of two segments, space and ground. This paper focuses on algorithm design for product generation within the ground segment. The instruments in the space segment will significantly increase the resolutions and therefore the data processing needs of the space and ground segments over the current generation instruments. Consider one instrument, the Advanced Baseline Imager (ABI). The ABI will increase the spectral resolution of GOES by three times over the exiting baseline imager. The spatial resolution will increase fourfold, and the temporal resolution will increase by half an order of magnitude. GOES-R will execute six L1 radiometric calibration algorithms, at least two image navigation and registration algorithms, and with Option 2, L2+ algorithms will generate over forty data products. This complexity demands strong software engineering to support both the infrastructure and the science algorithms to run operationally. L1b and L2+ algorithms will plug into a modular, scalable framework using data objects via the Data Model Interface (DMI). The DMI is designed using object oriented techniques to provide an abstract interface between internetworking of the infrastructure and the algorithms (see Figure 1).

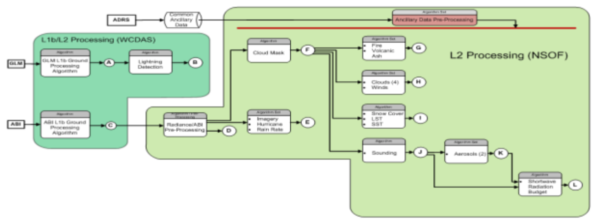


Figure 1: GOES-R Algorithm Execution

3. SCOPE OF THE STRUCTURED ANALYSIS APPROACH

Two software design approaches were identified during the proposal phase for use in the GOES-R Ground Segment (GS) development. Unified Modeling Language (UML) is the current de facto software design standard for object oriented development and was chosen for designing the overall Ground Segment software. Object oriented programming aids in developing complex software systems by promoting modularity and code reuse. This approach helps to decouple system complexity from subsystem complexity, therefore aiding in cost and risk reduction over non object oriented systems such as legacy monolithic architectures.

A structured analysis and structured design approach was specified for algorithm development. Two key rationales drive this approach. First, object oriented design is not a natural tool for analyzing and initially describing functional systems, such as satellite data processing algorithms. These algorithms are more easily analyzed and described through functional decomposition rather than through object creation. Second, physical scientists are not normally trained in object oriented design and are not expected to be familiar with tools such as UML. They are more likely to pick up such diagramming techniques as flowcharts and data flow diagrams.

Many analysis and design conventions had been developed for structured programming prior to the growth in popularity of object oriented design and programming. At the beginning of GOES-R development we needed to solidify our algorithm development approach and incorporate it into our formal procedures. We reviewed different structured approaches and chose SADT primarily for its terse symbol set, hierarchical diagramming approach, and readability. In order to comply with the overall GOES-R GS program development needs and achieve effective algorithm development within the program, we adopted key concepts of SADT instead of adopting the methodology wholesale.

Algorithm design fits within the larger context of an object oriented system (See Figure 2). As GOES-R is a large complex system, effective incorporation of a structured analysis algorithm design approach into the object oriented infrastructure design facilitates overall system integration.

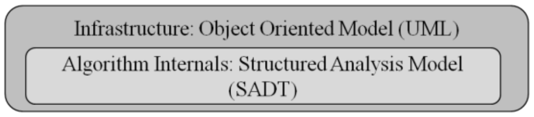

Infrastructure: Object Oriented Model (UML)
Algorithm Internals: Structured Analysis Model (SADT)

Figure 2: Algorithm design fits within the overall context of the infrastructure design

When the structured analysis and design approach was initially developed for GOES-R algorithm design the only UML artifacts specified were use cases (See Figure 3). Use cases were included because they link the software requirements to the design. No other object oriented design artifacts were going to be used as there is a clear separation between algorithm development and infrastructure development. The initial diagrams to augment SADT were flowcharts and entity relationship diagrams to provide control flow descriptions and structural models respectively. IBM Rational Software Architect is a UML design tool. It is used for the overall GOES-R Ground Segment software design. Therefore we had an opportunity to experiment with it for algorithm design; initially with class diagrams to replace entity relationship diagrams, then with activity diagrams to replace flowcharts. We found that this improved the quality of our design and our development processes for three reasons. First, it provides more consistency with the infrastructure design which fosters better communication. Second, there is a rich pool of information on the Internet and in reference books for UML design. Third, it allows greater integration of algorithm design with the GOES-R Ground Segment infrastructure software development processes.

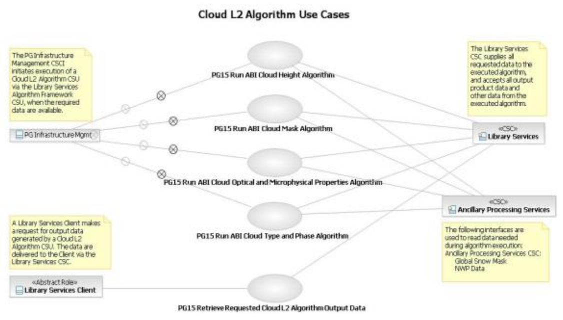

Cloud L2 Algorithm Use Cases
The PG Infrastructure Management CSCI initiates execution of a Cloud L2 Algorithm CSU via the Library Services Algorithm Framework CSU, when the required data are available.
PG Infrastructure Mgmt
PG15 Run ABI Cloud Height Algorithm
PG15 Run ABI Cloud Mask Algorithm
PG15 Run ABI Cloud Optical and Microphysical Properties Algorithm
PG15 Run ABI Cloud Type and Phase Algorithm
PG15 Retrieve Requested Cloud L2 Algorithm Output Data
The Library Services CSC supplies all requested data to the executed algorithm, and accepts all output product data and other data from the executed algorithm.
«CSC» Library Services
«CSC» Ancillary Processing Services
The following interfaces are used to read data needed during algorithm execution: Ancillary Processing Services CSC: Global Snow Mask NWP Data
A Library Services Client makes a request for output data generated by a Cloud L2 Algorithm CSU. The data are delivered to the Client via the Library Services CSC.
«Abstract Role» Library Services Client

Figure 3: UML use cases are drivers to tie design elements to requirements

4. STRUCTURED ANALYSIS AND DESIGN TECHNIQUE (SADT)

SADT is both a language and a process for systems analysis (Marca, 1991 p. 94). It is a functional modeling method for describing systems as a hierarchy of functions, and was chosen for algorithm development because it provides an unambiguous and relatively concise framework for systems modeling. For the GOES-R algorithm development effort, the primary goal of the SADT model is to develop correct descriptions of the science algorithms. An SADT model can focus either on system *things* or on system *activities*; historically the former are called “data models” and the latter “activity models”. An SADT activity model is similar to a data flow diagram but communicates significant additional information, in particular system *constraints*. The modeling process employs the classic top-down, hierarchic, functional decomposition approach to understanding and describing a system and to successively decomposing it into smaller and smaller subsystems, each of which can be described by an SADT diagram. The methodology is rigorous, with precise diagram syntax rules that enable an SADT model to concisely and accurately communicate significant information about the functionality and external interfaces of a system.

SADT artifacts differ from traditional analysis modeling artifacts. Traditional structured analysis relies on different diagram types, such as data flow diagrams and entity relationship diagrams, to describe a system. Fundamentally, SADT uses a single diagramming technique with a single element (shape). This fundamental diagram element is shown in Figure 4.

The function box represents a unit of activity: a process or action such as an equation or collection of decision logic. The function box has four arrows. Three arrows enter the box and one exits the box. Entering on the left of the box are the input data. These are data consumed by the function. Exiting from the right of the box are the output data. These are data produced by the function.

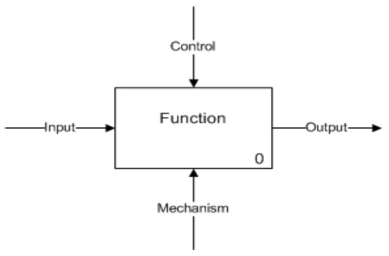


Figure 4: The SADT function box

The basic elements involved in the development of an SADT diagram are as follows**:**

- **The Box –** The function, action, or process (the *activity*)
- **Input –** Data consumed by the function
- **Output –** Data produced by the function
- **Control –** Rules or constraints on the activity, which influence the execution of the activity
- **Mechanism –** The context of the activity, i.e. the means of realizing the activity defined by the box. This is a convenient hook to apply classes or UML use case actors to SADT model behavior

The key general concepts that govern the construction of any SADT diagram are:

- The function represented by the box consumes input and produces output based upon conditions of the controls and application of the mechanism to the function
- SADT arrows represent a set of one or more *things* (such as numerical data sets)
- SADT arrows are *decomposable*, i.e. they can split and join to indicate separation or aggregation of the represented data sets
- SADT, unlike other structured analysis methods, distinguishes input data from control data. Inputs can be considered as things a function consumes; controls define rules or constraints on a function

All diagram elements (boxes, arrows, and interfaces) are uniquely labeled, enabling unambiguous association of an activity box and its interfaces with its decomposed sub-diagram. This diagramming approach readily lends itself toward satellite data processing algorithms, which follow an input → processing → output behavioral model (See Figure 5). In the context of GOES-R science algorithms, the input and output arrows correspond to time series and location based data where inputs are for a value or range of time T and location L. The control data are generally considered constant over a span of inputs.

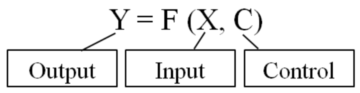


Figure 5: Function as a basis for the SADT function box

SADT has a specific diagram hierarchy organized into three groups:

- System context diagram
- System summary diagram
- N tier decomposition

At the top is the system context diagram. This diagram identifies all the external interfaces to the algorithm (the system). The system context diagram provides a top-level/abbreviated view of the SADT model. Using SADT nomenclature, this box is labeled A-0 and is at the top of the hierarchy (See Figure 6). A single box is shown in this diagram to represent the entire algorithm. This diagram thus treats the complete algorithm as a "black box" and emphasizes the external interfaces of the algorithm, including the input required and the output produced. In SADT notation, a hyphen is used between the letter and the number for the system context diagram.

The next SADT diagram is the system summary diagram, labeled A0. This diagram provides the first-level decomposition of the constituent activities internally performed by the system and the internal flow of data that occurs between these activities in order to accomplish the system's purpose. The context (A-0) and summary (A0) diagrams operate as a pair where: (1) the context diagram provides the top level "black box" view of the algorithm, and (2) the summary diagram provides the first level "white box" decomposition of the three to six major components within the algorithm. Each box within a diagram is assigned and labeled with a number that is unique within the diagram, starting with "1" and continuing sequentially. Sub-diagrams are developed from each box in the A0 diagram. Each sub-diagram of A0 is labeled A1, A2, A3, and so on (See Figure 6). Further decomposition of each function box into a lower level diagram in the hierarchy appends this box's number to the diagram label. For example, the sub-diagram of box 1 of the A0 diagram will be A1. The sub-diagram of box 3 of the A1 diagram will be labeled A13 (See Figure 6).

Excluding the system context diagram (A-0), SADT prescribes that each other activity diagram, including the system summary (A0) diagram, contain between three and six boxes describing the constituent sub-activities. In GOES-R algorithm development, we have found through practice that there are cases where deviating from the six box limit results in cleaner grouping of design elements. Therefore, we use the three to six box range as a general guide rather than adhere to it strictly.

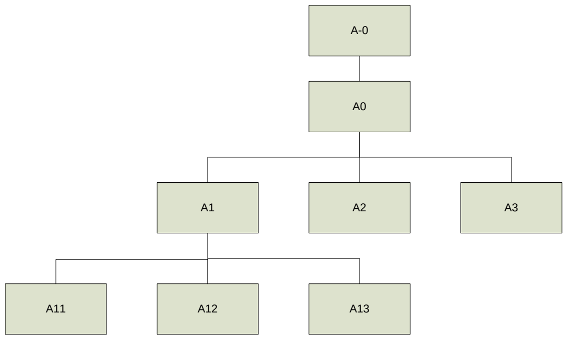


Figure 6: Representation of an SADT diagram hierarchy

5. STRENGTHS AND DRAWBACKS OF SADT

5.1. Strengths of SADT

In GOES-R, SADT has provided a consistent modeling tool to describe how the GOES-R algorithms work as systems. Spatially and temporally dependant data flows are differentiated from static and semi-static data. Algorithms in general, and science algorithms in particular, follow an input → processing → output behavioral model. This is readily represented with SADT. This approach has benefits of consistency and adaptability and has proven effective in GOES-R algorithm development for communicating the correctness of algorithms.

- SADT uses a single building block; the SADT function box. This promotes understanding by avoiding the use of a variety of diagramming techniques to describe system hierarchies
- SADT provides an unambiguous and relatively concise framework for systems modeling
- SADT is easily complemented with other techniques
- Physical scientists and software engineers can use SADT as a common language to ensure design correctness

5.2. Drawbacks of SADT

SADT is not a design magic bullet. It addresses many, but not all, analysis and design issues. It also has some inherent flaws. Two key inherent flaws are:

- Mechanism – This is the least well defined part of the SADT function box and has caused confusion as to its purpose and usefulness
- Decisions – SADT lacks the definition of control flow that is present in flowcharts and UML activity diagrams

Another issue with SADT is its lack of readily available design tools. GOES-R uses Visio 2007, which is time consuming for creating and modifying diagrams which contain many data flow lines (arrows). In contrast, there are many commercial and free tools available for UML. SADT describes how a hierarchical system works, which lends well toward procedural programming, but does not define how the system should work as object oriented code. These drawbacks, along with the benefits of closer integration with infrastructure design, were driving forces for augmenting SADT with UML.

6. UNIFIED MODELING LANGUAGE (UML)

UML is a standardized modeling language for developing object oriented software systems. It uses a collection of diagrams grouped into two major categories: behavioral diagrams and structural diagrams. Behavioral diagrams describe how a system being modeled must operate. Structural diagrams describe the entities that must be in a system being modeled. GOES-R relies heavily on a subset of UML diagrams. Behavioral diagrams used are:

- **Use Case diagrams -** describe actor-based functionality of a system through use case analysis
- **Activity diagrams** - describe conditional flow and state behavior
- **Sequence diagrams -** describe temporal flow and object lifecycles

For structural diagrams, GOES-R uses class diagrams to describe classes, attributes, and relationships between classes.

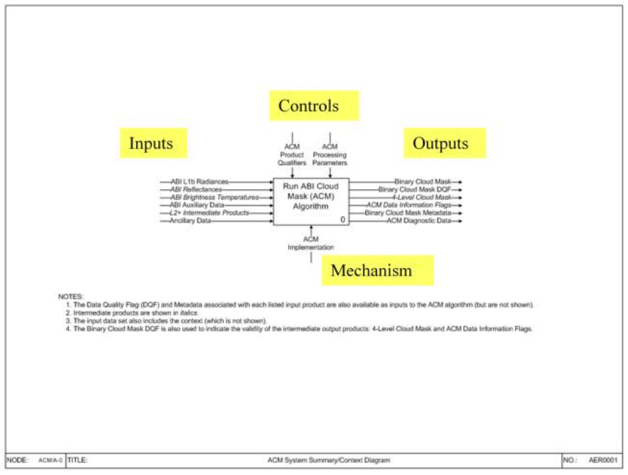
7. AUGMENTING SADT WITH UML

UML solves key issues of SADT by providing:

- A means to describe control flow via UML activity diagrams
- Sequence diagrams to identify timing and software component lifecycles
- Class diagrams to provide structures in an object oriented manner

The following diagrams are from the design of the Advanced Baseline Imager (ABI) Cloud Mask algorithm. They provide an illustration of the system decomposition. Figure 7 shows the system context diagram which provides a “black box” view of the external interfaces to the algorithm. Figure 8 shows the system summary diagram which provides a first level decomposition of the major functional components within the Cloud Mask algorithm. Figures 9 and 10 provide the second and third tier decompositions of box 2 (binary cloud detection tests) of the system summary diagram.

The UML diagrams in this section show capabilities which SADT diagramming lacks.



NOTES:
1. The Data Quality Flag (DQF) and Metadata associated with each listed input product are also available as inputs to the ACM algorithm (but are not shown).
2. Intermediate products are shown in italics.
3. The input data set also includes the context (which is not shown).
4. The Binary Cloud Mask DQF is also used to indicate the validity of the intermediate output products: 4-Level Cloud Mask and ACM Data Information Flags.

Figure 7: SADT context diagram for the ABI Cloud Mask algorithm. Note the label “A-0 in the lower left corner”

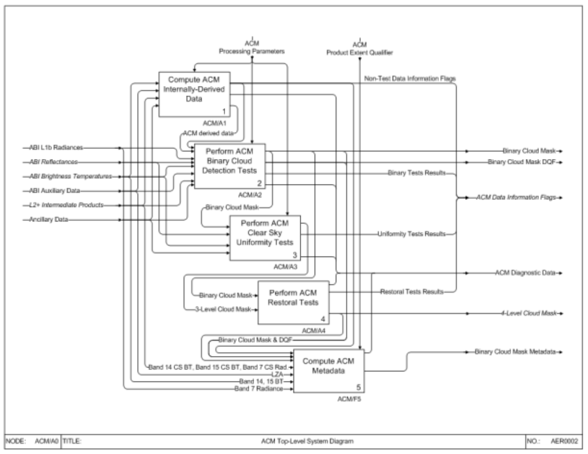


Figure 8: SADT system summary diagram for the ABI Cloud Mask algorithm. Note the label "A0 in the lower left corner"

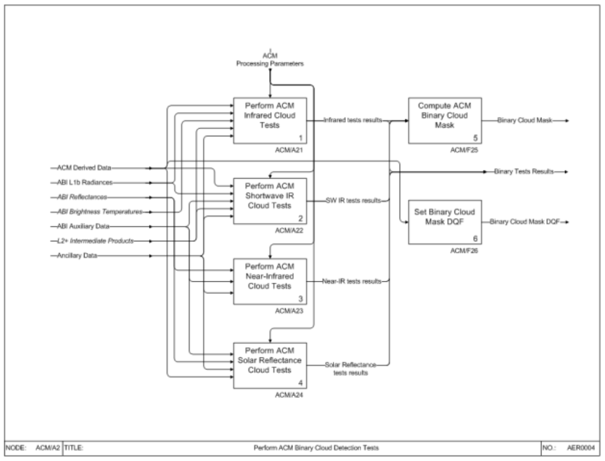

ACM Processing Parameters
Perform ACM Infrared Cloud Tests 1 ACM/A21
Perform ACM Shortwave IR Cloud Tests 2 ACM/A22
Perform ACM Near-Infrared Cloud Tests 3 ACM/A23
Perform ACM Solar Reflectance Cloud Tests 4 ACM/A24
Compute ACM Binary Cloud Mask 5 ACM/F25
Set Binary Cloud Mask DQF 6 ACM/F26
ACM Derived Data
ABI L1b Radiances
ABI Reflectances
ABI Brightness Temperatures
ABI Auxiliary Data
L2+ Intermediate Products
Ancillary Data
Infrared tests results
SW IR tests results
Near-IR tests results
Solar Reflectance tests results
Binary Cloud Mask
Binary Tests Results
Binary Cloud Mask DQF
NODE: ACM/A2 TITLE: Perform ACM Binary Cloud Detection Tests NO.: AER0004

Figure 9: SADT A2 diagram for the ABI Cloud Mask algorithm which shows the decomposition of box 2 of the system summary diagram (A0), "Perform Binary Cloud Detection Tests"

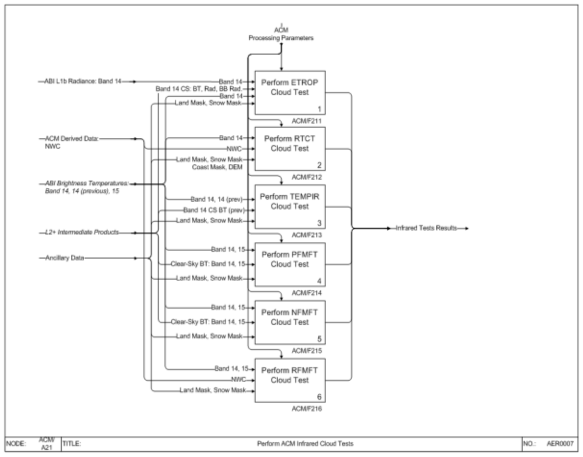

ACM Processing Parameters
ABI L1b Radiance: Band 14
ACM Derived Data: NWC
ABI Brightness Temperatures: Band 14, 14 (previous), 15
L2+ Intermediate Products
Ancillary Data
Perform ETROP Cloud Test 1 ACM/F211
Perform RTCT Cloud Test 2 ACM/F212
Perform TEMPIR Cloud Test 3 ACM/F213
Perform PFMFT Cloud Test 4 ACM/F214
Perform NFMFT Cloud Test 5 ACM/F215
Perform RFMFT Cloud Test 6 ACM/F216
Infrared Tests Results
NODE: ACM/A21 TITLE: Perform ACM Infrared Cloud Tests NO.: AER0007

Figure 10: SADT A21 diagram for the ABI Cloud Mask algorithm which shows the decomposition of box 1 of the A2 diagram, "Perform ACM Infrared Cloud Tests"

A UML activity diagram describes the control flow logic of a component or set of components. Like flowcharts, rectangles identify activities and diamonds represent decisions. Each component's responsibilities are distinguished by a vertical swim lane. Graphically, swim lanes are vertical or horizontal rectangular regions spanning the height or width of the diagram respectively. They identify who is supposed to perform which activity and make which decision. If there is only one component, then that component's swim lane takes up the entire diagram. Figure 11 shows the control flow logic of the SADT diagram A21 box 5, "Perform NFMFT Cloud Test." This diagram shows that the algorithm component on the left swim lane is executing a "runCheck" method of the NFMFT Cloud Test design element and implemented by the `NFMFT_Check` Class.

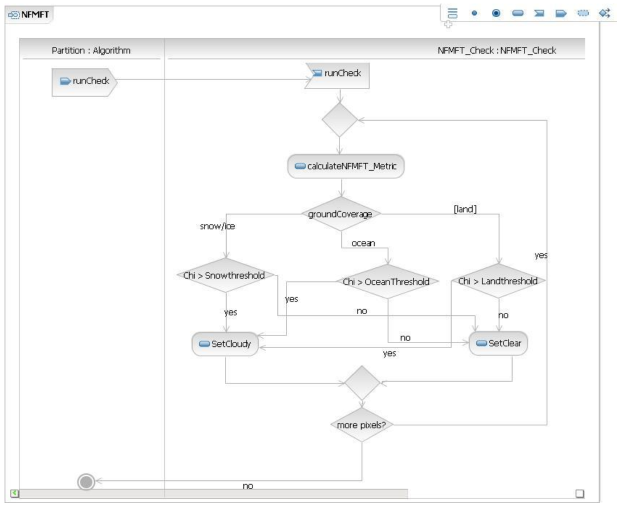


Figure 11: UML Activity Diagram

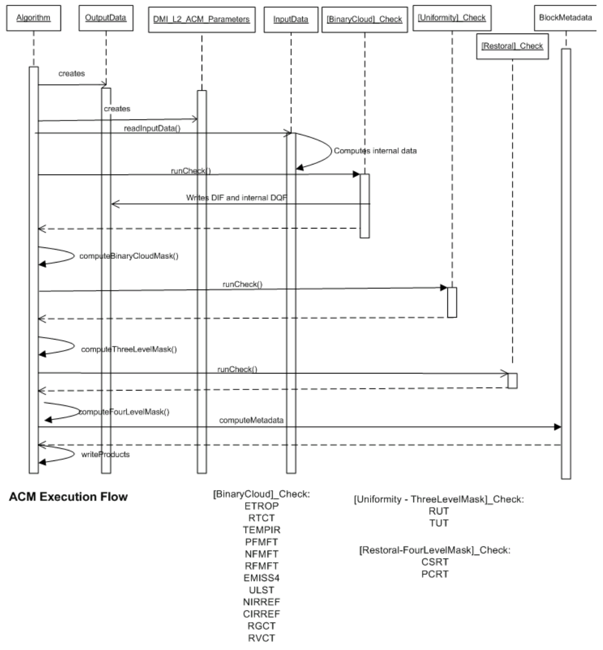


Figure 12: UML sequence diagram describes a sequence of executions needed to fulfill some tests to create the cloud mask product data of the ABI Cloud Mask algorithm

A UML Sequence diagram describes temporal flow and object lifecycles (see Figure 12). Each process or object is identified by a box on the top of the diagram. Each vertical bar below the object box identifies the object's lifeline, which its scope of execution. Arrows are messages and function calls between objects. This diagram helps to discover and address timing issues. For GOES-R algorithm development, they help develop class composition and integration with the service framework.

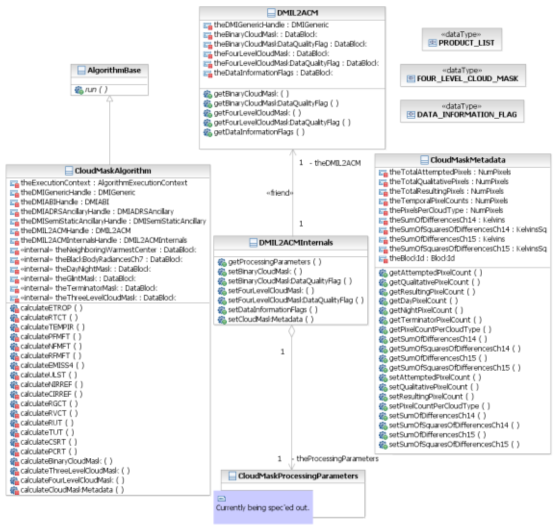

AlgorithmBase
run ( )
DMIL2ACM
theDMIGenericHandle : DMIGeneric
theBinaryCloudMask : DataBlock
theBinaryCloudMaskDataQualityFlag : DataBlock
theFourLevelCloudMask : DataBlock
theFourLevelCloudMaskDataQualityFlag : DataBlock
theDataInformationFlags : DataBlock
getBinaryCloudMask ( )
getBinaryCloudMaskDataQualityFlag ( )
getFourLevelCloudMask ( )
getFourLevelCloudMaskDataQualityFlag ( )
getDataInformationFlags ( )
«dataType» PRODUCT_LIST
«dataType» FOUR_LEVEL_CLOUD_MASK
«dataType» DATA_INFORMATION_FLAG
CloudMaskAlgorithm
theExecutionContext : AlgorithmExecutionContext
theDMIGenericHandle : DMIGeneric
theDMIABIHandle : DMIABI
theDMIADRSAncillaryHandle : DMIADRSAncillary
theDMISemiStaticAncillaryHandle : DMISemiStaticAncillary
theDMIL2ACMHandle : DMIL2ACM
theDMIL2ACMInternalsHandle : DMIL2ACMInternals
«internal» theNeighboringWarmestCenter : DataBlock
«internal» theBlackBodyRadiancesCh7 : DataBlock
«internal» theDayNightMask : DataBlock
«internal» theGlintMask : DataBlock
«internal» theTerminatorMask : DataBlock
«internal» theThreeLevelCloudMask : DataBlock
calculateETROP ( )
calculateRTCT ( )
calculateTEMPIR ( )
calculatePFMFT ( )
calculateNFMFT ( )
calculateRFMFT ( )
calculateEMISS4 ( )
calculateULST ( )
calculateNIRREF ( )
calculateCIRREF ( )
calculateRGCT ( )
calculateRVCT ( )
calculateRUT ( )
calculateTUT ( )
calculateCSRT ( )
calculatePCRT ( )
calculateBinaryCloudMask ( )
calculateThreeLevelCloudMask ( )
calculateFourLevelCloudMask ( )
calculateCloudMaskMetadata ( )
1 - theDMIL2ACM
«friend»
1
DMIL2ACMInternals
getProcessingParameters ( )
setBinaryCloudMask ( )
setBinaryCloudMaskDataQualityFlag ( )
setFourLevelCloudMask ( )
setFourLevelCloudMaskDataQualityFlag ( )
setDataInformationFlags ( )
setCloudMaskMetadata ( )
1
1 - theProcessingParameters
CloudMaskProcessingParameters
Currently being spec'ed out.
CloudMaskMetadata
theTotalAttemptedPixels : NumPixels
theTotalQualitativePixels : NumPixels
theTotalResultingPixels : NumPixels
theTemporalPixelCounts : NumPixels
thePixelsPerCloudType : NumPixels
theSumOfDifferencesCh14 : Kelvins
theSumOfSquaresOfDifferencesCh14 : KelvinsSq
theSumOfDifferencesCh15 : Kelvins
theSumOfSquaresOfDifferencesCh15 : KelvinsSq
theBlockId : BlockId
getAttemptedPixelCount ( )
getQualitativePixelCount ( )
getResultingPixelCount ( )
getDayPixelCount ( )
getNightPixelCount ( )
getTerminatorPixelCount ( )
getPixelCountPerCloudType ( )
getSumOfDifferencesCh14 ( )
getSumOfSquaresOfDifferencesCh14 ( )
getSumOfDifferencesCh15 ( )
getSumOfSquaresOfDifferencesCh15 ( )
setAttemptedPixelCount ( )
setQualitativePixelCount ( )
setResultingPixelCount ( )
setPixelCountPerCloudType ( )
setSumOfDifferencesCh14 ( )
setSumOfSquaresOfDifferencesCh14 ( )
setSumOfDifferencesCh15 ( )
setSumOfSquaresOfDifferencesCh15 ( )

Figure 13: ABI Cloud Mask Algorithm Major Classes

Class diagrams are developed in concert with expanding the SADT model and updated as design efforts progress. The following diagram (Figure 13) shows the major classes in the ABI Cloud Mask algorithm. A class is a static structure which defines a collection of methods and attributes (data members). A class defines the smallest granularity of functionality in an object oriented system and defines the interface for this functionality. In this way, classes can be thought of as templates or types from which objects are instantiated to perform computation to fit the logic specified by the class. A class diagram identifies the static structure of a whole or part of an object oriented system. This diagram identifies classes and their relationships, such as inheritance and association.

8. CONCLUSION

SADT has key significant strengths which make it an effective and appropriate tool for graphically describing and designing data processing systems in general and algorithms with rich hierarchies of equations in particular. First, SADT diagramming has a low learning curve both for those creating and reading the diagrams. The diagrams are built upon a single building block, the function box. Once one understands the function box, then the diagram organization should be clear. Second, SADT differentiates between dynamic and static data. If correct diagramming techniques are followed, then the reader can immediately recognize which data are dynamic, such as time series and location range and which data are static, such as configuration parameters and coefficients. Each diagram below the context diagram should have as a general rule between three to six function boxes, which promotes a concise and uncluttered reading of a set of functionality.

SADT has drawbacks too. It does not describe how software should be implemented in an object oriented way. The hierarchical decomposition lends readily to implementation with a set of functions, but does not clearly show how those functions can be organized in effective manner of cohesion into classes. This requires knowledge of object oriented design to map function boxes into classes and class methods. SADT does not describe state nor provide for control flow. It also does not describe component lifecycles or timing behavior of components.

Augmenting SADT with UML diagrams fills the gaps in SADT as a software design tool. UML activity diagrams describe the control flow within SADT function boxes. UML sequence diagrams describe temporal flow and object lifecycles. In GOES-R, SADT has proven effective as a shared language between scientists, systems engineers, and software engineers to demonstrate algorithm correctness. UML is an industry standard object oriented design approach and provides a consistent language for developing object oriented systems. When combined, these methodologies have demonstrated a consistent and effective design approach toward developing GOES-R algorithms.

9. REFERENCES

Dickover Melvin E., McGowan Clement L. and Ross Douglas T., 1977: Software Design using SADT [Conference] // ACM Annual Conference/Annual Meeting.

Marca David A. and McGowan Clement L., 2006: IDEF0 and SADT, A Modeller's Guide [Book]. - Auburndale, MA : OpenProcess, Inc.