NAVIGATION AND MISSION ANALYSIS SOFTWARE FOR
THE NEXT GENERATION OF JPL MISSIONS

Steven FLANAGAN
Todd ELY

Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
Steve.N.Flanagan@jpl.nasa.gov

ABSTRACT – MONTE (Mission analysis, Operations, and Navigation Toolkit Environment) is a new software system being developed to replace the navigation and trajectory analysis software currently in use at the Jet Propulsion Laboratory (JPL). MONTE will reproduce the existing functionality of the legacy systems and add significant new capabilities for the MONTE users - the mission and navigation analysts in the Navigation and Mission Design Section (Section 312). MONTE will be developed as a single tightly integrated system, in contrast to the multiple disparate software suites currently in use. MONTE is being designed to facilitate a variety of navigation and trajectory tasks in a broad range of contexts, including research and development, analysis and design, and operations.

KEYWORDS: Navigation, mission design, operations, MONTE, software, system, trajectory.

INTRODUCTION
MONTE (Mission analysis and Operational Navigation Toolkit Environment) is a new software system being developed to support trajectory and navigation analysis/design for space missions. MONTE will eventually replace the current navigation and trajectory analysis software systems used by the Navigation and Mission Design Section (Section 312) of the Jet Propulsion Laboratory (JPL). MONTE will have the capabilities of these existing systems, plus add significant new tools for use by mission and navigation analysts. Furthermore, it does so within the framework of a single, integrated system. It has a variety of user interaction modes ranging from traditional command-driven, standalone applications, to a graphics-based interactive computing environment. MONTE is primarily a ground-based system; however, it is likely that future adaptations of MONTE components will be used as embedded parts of a flight system.

MONTE supports a variety of navigation and trajectory tasks. Fundamentally, it has been designed and built to be useful in a broad range of contexts, including research and development, analysis and design, and operations. Several examples of how MONTE can be used include,

- Plan and design spacecraft trajectories that satisfy mission objectives,
• Estimate and control the actual spacecraft trajectories throughout the mission,
• Maintain and disseminate knowledge of the spacecraft trajectories,
• Provide related information for solar system bodies such as planetary orbits, pole orientations and rotation rates, etc.
• Develop new algorithms for trajectory targeting; conduct feasibility studies on new orbit determination techniques, etc.

Note that the MONTE system will be capable of supporting missions with multiple spacecraft as well as the more traditional interplanetary mission with only a single spacecraft.

It is reasonable to ask whether the expense of developing a new software system such as MONTE is justified at this time. In what respects will MONTE represent an improvement over the existing navigation and mission design software tools? In addition to the significant new functionality that is being added to the Navigation and Mission Design Section's core capabilities, MONTE is also developing a system that adheres to the following principles:

• MONTE must be easy to use for both the novice and the seasoned practitioner,
• MONTE must be flexible so that it can be easily adapted for use on new navigation and mission design applications,
• MONTE must seamlessly integrate the related, yet distinctly different, functionality required by mission designers and navigators,
• MONTE must be measurably more stable and easier to maintain in an environment where many and varied missions are the norm, not the exception.

MONTE PROJECT OBJECTIVES

The fundamental task of the MONTE Project as originally conceived is to functionally replace JPL’s existing legacy navigation software system known as DPTRAJ/ODP. This is to be accomplished through the development of a new software system that will address the known shortcomings of the existing software, as opposed to a line-by-line or function-by-function rewrite of the legacy code. The scope of the MONTE Project has been expanded to encompass Mission Analysis software as well. This includes the legacy Mission Analysis Software Libraries (MASL) and several new software developments either planned or already under way in this area. In replacing these legacy systems, MONTE has the challenge of selecting the architecture that will achieve the most significant improvements in all aspects of the software. In order to accomplish this goal, it is necessary to identify the highest priority objectives for this project.

Reduce Life-cycle Cost

One of the strongest arguments in favor of pursuing this development effort is that by doing so, it is possible to reduce the overall lifecycle cost associated with maintaining the legacy software. Maintaining the legacy software as robust, reliable software systems that meet the needs of their users is an expensive proposition, in part due to the fact that they are based on aging technology, and also because of the way these systems have evolved over time. Building new software from the ground up is expected to result in a system that is easier, and therefore less expensive, to maintain, without sacrificing functionality. One key element of this plan is to provide accurate and current documentation of software requirements, design, and code for all software components in the MONTE system.

Improve Usability

Most of the legacy software currently in use provides Command-Line User Interfaces (CLUIs) exclusively. The appropriate application of Graphical User Interfaces (GUIs) will significantly enhance
the user-interaction characteristics of the new software. Well-designed and constructed GUIs will make
the software more accessible to the novice, without getting in the way of the expert. Standardized GUIs
will also make it easier for the Navigation and Mission Design Section or for a flight project to provide
officially approved approaches to using the software.

It would be inappropriate, however, to assume that GUIs are preferable to CLUIs in all applications. The
prospective users of MONTE have repeatedly voiced an interest in having alternatives to GUIs.
Therefore, another user-interface goal of MONTE is to provide a common interactive command-line and
scripting capability to both Navigation and Mission Analysis users.

**Provide Enhanced Functionality**

The domains of Navigation and Mission Analysis are dynamic, and it is reasonable to expect that
MONTE’s user requirements will change somewhat over time. As a result, it will never be possible to
anticipate all potential uses of the software at the time of design and implementation. Many new
requirements have come out of the demanding missions attempted by JPL in the years since the legacy
software was originally developed. This situation has challenged Section 312’s ability to maintain and
extend these software sets. MONTE must address the additional requirements expected based on our
present knowledge of the missions in NASA’s roadmap and be able to respond readily to new
requirements that cannot be anticipated. This can only be accomplished through the careful design and
documentation of an extensible and maintainable software system.

**Integrate Mission Analysis and Navigation Software Systems**

In spite of the functional overlap between Navigation and Mission Analysis, these two areas have always
maintained separate core software libraries at JPL. This has introduced a number of problems, with the
two most important being maintenance cost and interoperability. Maintaining two software systems with
very similar functional requirements is clearly more expensive than maintaining a single library.
Furthermore, it is difficult to find new people with the necessary FORTRAN skills. Finally, it is very
common for Mission Analysts to deliver solutions to Navigation Analysts. This is currently a complicated
process that can contribute to errors being made in the translation from one software set to another. This
situation is unnecessary. Building a single library that can be shared by these two areas will introduce
economies and efficiencies that cannot be realized any other way.

**KEY HIGH-LEVEL FUNCTIONAL REQUIREMENTS**

As mentioned previously, the primary functions of the MONTE system are to analyze and design
spacecraft trajectories; estimate and control spacecraft trajectories; and maintain and disseminate
knowledge of spacecraft trajectories and related information for solar system bodies. MONTE can be
used during all spacecraft mission phases from pre-project status through final mission operations.
Furthermore, MONTE can be used for research and technology studies not necessarily associated with a
specific mission.

A significant feature associated with MONTE is that all of its diverse functionality resides within the
context of a single, integrated system. The immediate implication of this is that MONTE is capable of
servicing all mission design and navigation functions seamlessly throughout a mission lifecycle. This
reduces the potential for error and yields efficiencies that are currently not available with existing mission
design and navigation software systems. For example, the models, data, and trajectory products that a
mission designer generates during initial design studies are the same products that a navigator can use for
initial covariance studies and to define DSN tracking requirements. During operations, the same detailed
spacecraft models used for operational navigation support can be used, without modification, for
trajectory redesign activities as the need may arise.

This seamless operability applies not only to models and data products, but to the MONTE tools, as well.
For instance, analysis and design are not restricted to the pre-launch time frame. MONTE tools allow the
user to evaluate alternatives for future mission phases while supporting on-going operations for the
current mission phase. The same estimation tools used to obtain a trajectory solution based on the latest
actual measurements are also used to perform covariance studies using a simulated set of future
measurements. Similarly, the same trajectory design tools used to generate a nominal mission trajectory
can be used to re-plan or re-optimize remaining future trajectories based on the actual past trajectory
history or in response to unexpected perturbations from the nominal.

The tasks that MONTE can do fall into the following major functional areas:

- **Trajectory and Variations Generation** contains components that generate spacecraft or celestial
  body trajectories via an analytical formulation or numerical integration of a dynamic model. It also
  generates the variations of dynamic variables and dynamic model parameter quantities at requested
time intervals (i.e., the state transition matrix).

- **Trajectory Design and Control** contains components that manipulate trajectories to achieve
  certain desired conditions via targeting or optimization. This includes maneuver analysis functions
  for statistical assessment of control events.

- **Trajectory Information** contains components for disseminating trajectory knowledge. This
  includes tools for accessing basic trajectory representations and associated model parameters and
  for computing numerous geometric quantities that can be derived from the trajectory.

- **Measurement Scheduling, Simulation, Variations, and Acquisition** contains components for
  modeling measurement schedules, simulating measurements, generating measurement variations
  with respect to dynamic variables and model parameters. There are tools to acquire measurements
  (via a "real-time" stream, or as a batch file), process them (i.e., sampling and/or filtering, applying
  calibration corrections, etc), creating derived measurement types and processing them, as well.

- **Trajectory and Parameter Estimation** contains components that solve for spacecraft or celestial
  body trajectories and associated model parameters based on a series of simulated or actual
  measurements. This includes various types of filtering and smoothing algorithms and tools to
  manipulate and map state uncertainties and other products of the filtering process.

- **Textual and Graphical Data Analysis** contains tools that have significant capabilities to analyze
  and visualize MONTE generated data. This includes text-based processing (i.e., convenient output
  forms for viewing and porting data to 3rd party spreadsheet tools) and visualization of output data
  (and in some cases input data). Almost all navigation and trajectory analysis tasks will have
  associated visualization techniques that are useful in performing the task or evaluating the results.
  Selected examples of 2D/3D visualization capabilities include:
  - Trajectory plots of spacecraft and celestial bodies;
  - 3-D displays of spacecraft component models;
  - Coordinate system axes definitions and their relation to other systems;
  - “Function graphs” of one or more parameters as functions of time or another parameter, e.g. a
    plot of flight path angle versus magnitude of the B-vector at atmospheric entry interface;
  - Timelines of measurement opportunities and associated detailed schedules;
  - Actual and simulated measurement residuals;
  - Display raw or transformed uncertainties, e.g. a 2D plot of the B-plane error ellipse at closest
    approach to a target body.

When appropriate, the user will have the ability to interact with their specific analysis scenario (i.e.,
direct changes to parameters and data) via manipulation of information in the graphics.
• **Case Management** contains tools for managing team-based mission and navigation analysis and design. They manage user environments, scripts designed for analysis and task automation, data consisting of inputs as well as outputs, coordinating team member activities. It can be thought of as a data and task management function that ties together the previous functional blocks with mission data/results that need to be shared, managed, and configuration controlled. A primary objective of this tool is to ensure data integrity of operational mission products.

• **Infrastructure** contains lower-level tools necessary to accomplish the preceding functions. It includes capabilities to process and manage trajectory components, time systems, coordinate systems, mathematics, data management, etc. Of course, there are software tasking capabilities that coordinate the execution of all the functionality, ensuring that processing proceeds as expected and provides the user with requested data/analyses or notifies them of error conditions preventing normal task completion.

These functional areas represent a loose partitioning of the MONTE system, hence the idea that MONTE is a toolkit. From the toolkit perspective, there are four primary ways to utilize the MONTE system. MONTE, used as an *interpreter*, enables the user to call functions to do specific mission design and navigation tasks. The MONTE system can also be viewed as a set of *pre-built applications* that seamlessly tie toolkit functions together and enable a user to easily accomplish a complex series of standard tasks. MONTE's scripting capability allows the user to create new *script-based applications* that can integrate functionality from the toolkit layer, pre-built applications, and/or other software developed outside of the MONTE environment. Finally, MONTE can be used as a *software library* to build new compiled applications.

**MONTE SOFTWARE DEVELOPMENT APPROACH**

MONTE is a large scale, long-term software project that must be developed and maintained by a team in Section 312. The software products of MONTE will be used to design and navigate future JPL missions, and will therefore be classified as “mission critical software”. This implies that a very high degree of reliability is required of the software. Versions of the current JPL navigation software have been in continuous use for nearly 35 years. Clearly, MONTE must plan on being highly robust and maintainable over a long period of time. Several aspects of the MONTE approach to software development that support these requirements are described in more detail in this section.

**Formal Software Process**

Although there are usually several software development efforts underway at any given time within the Navigation and Mission Design Section, software development is not the primary mission of the Section. Most of the software tasks in the Section are small and informal. This is clearly not appropriate for a project with the size and scope of MONTE. A great deal of effort has gone into providing a clear and stable set of software development guidelines, standard practices, policies, and procedures for the project. This is necessary to ensure that MONTE can demonstrate that the requirements are understood and are being addressed, that the development is being completed on schedule, and that the delivered products are of an appropriate quality. Documented procedures are in place or under development for activities covering the entire software lifecycle, including requirements analysis, software project planning, configuration management, and software quality assurance.

**Object-Oriented Design/C++**

In order to satisfy the design goals described earlier, MONTE must be a reliable, extensible, and reusable software system. These objectives impose requirements on the system that are not derived from the more obvious and well-understood technical software requirements. Although a comprehensive discussion of the benefits of object-oriented analysis and design is beyond the scope of this paper, it was decided that an object-oriented approach was most likely to succeed in addressing the broad objectives of MONTE.
This left several options for software languages. The three given the most serious consideration were Fortran 90, Java, and C++. The vast majority of existing software in Section 312 is written in one of several dialects of Fortran, including Fortran 77, Ratfor, and Fortran 90. As a result, a strong argument could be made for continuing this practice. Fortran 90 is a well-designed language, and high-quality, stable compilers are available for all of the platforms currently in use in the Section. Fortran 90 benefits from the algorithmic roots of Fortran, and is well suited for use in solving many of the highly mathematical problems in Navigation and Mission Design.

However, it is important to realize that MONTE is much more than a set of related algorithms. Significant effort will be directed towards developing graphical user-interfaces, which is not one of Fortran 90’s strengths. In addition, it is expected that some components of MONTE will evolve into on-board flight software at some point in the future. Although the current flight software at JPL is written in C, future flight software developments plan to phase in the use of C++. Fortran is not being considered for flight software, and is not supported by any of the on-board platforms used for flight systems. Finally, Fortran programmers are becoming very difficult to find. Fortran is not being taught in many Engineering schools, and very few Computer Science graduates have had any experience with it at all. This is an important limitation, and ultimately contributed to the decision not to use Fortran.

This left Java and C++ as viable options. Java is a true object-oriented language, and is gaining popularity rapidly. Many new graduates have only programmed in Java, and are excited about continuing to use it. However, the Java language is still fairly immature, and as a result, the standard is fairly unstable. This could be a very serious problem for a long-term, large-scale project. Also, the performance of Java doesn’t compare very favorably to C or C++. Taken together, these factors eliminated Java as an option. C++ is mature, stable, widely used, fully object-oriented, and has performance characteristics appropriate for MONTE’s applications.

**Toolkit Approach**

The phrase “Toolkit Approach” is used to describe the practice applied in the MONTE project of developing a large library of reusable software components with very broad capabilities in the functional areas of Navigation and Mission Design. This is an important part of the MONTE architecture, and delivers a number of benefits to the users of MONTE. This approach makes it possible for an analyst to repackage existing functionality to solve new kinds of problems. While this degree of flexibility might not be desirable in an operational scenario, it could be essential to support the wide variety of preliminary studies conducted in the Section. MONTE will make every effort to deliver a standard set of tools that meet the requirements of the Navigation and Mission Design users, but it is unreasonable to expect that MONTE will anticipate all of the ways that the delivered software can be used.

Also, it is unavoidable that at some point in the future additional user requirements will be levied on MONTE, possibly to support new classes of missions that are not currently anticipated. The toolkit approach promotes extensibility by providing a framework for adding new functionality to MONTE, without unnecessarily complicating the existing software. This is an important feature for a software system that could potentially be expected to survive for decades.

**MONTE ARCHITECTURAL DESIGN**

The MONTE system architecture can be best understood by thinking in terms of layers of functionality. As shown in Figure 1 there are four conceptual layers in the MONTE system: the Data Management Layer, the Toolkit Layer, the User Interface Component Layer, and the User Application Layer. The following sections describe the functionality provided by these layers and their roles within the MONTE system.
**Data Management Layer**

BOA, the MONTE data management system, represents the lowest level of functionality in the MONTE system. In large, complex, object-oriented software systems such as MONTE there is a need for object persistence. Object persistence is the ability for objects (instances of C++ classes) to survive beyond the lifespan of the component or application that created them. This capability directly enables code reuse by allowing applications to use objects that have been created elsewhere in the system. The primary function of BOA is to save and restore C++ objects in a machine portable format. BOA plays an important role in the architecture and use of the MONTE system, and its presence is likely to influence the design of most MONTE subsystems to some extent. As a result, it is important that software designers consider BOA in the design of any new MONTE subsystem, and that individuals responsible for designing major software components within MONTE understand the capabilities and proper uses of BOA.

**Toolkit Layer**

The core Mission Design and Navigation functionality is provided by the part of MONTE that is labeled the Toolkit Layer in Figure 1. This is a collection of software libraries that can be used together to address a wide range of problems within the target problem domain. These libraries provide many of the capabilities that are fundamental to the work performed in Section 312, including Trajectory Analysis, Measurement Analysis, Coordinate and Time System calculations, Estimation, Optimization, and others. These low-level functional libraries are designed to use the MONTE data management system, and can therefore take advantage of the services provided by BOA.

**User Interface Component Layer**

The users of MONTE cannot be expected to write a script or a C++ program every time they need to solve a problem. The functionality represented by the Toolkit Layer described above must be organized into applications, and presented to the users in a convenient and straightforward fashion. Towards this end, a collection of user interface components are planned that will facilitate the production of end-user applications based on the Toolkit Layer.

The user interface components fall into two categories: Graphical User Interface (GUI) components and Command-line User Interface (CLUI) components. Both styles of interaction are important to the users of MONTE, and must be supported. It is expected that nearly all end-user applications will provide users with a CLUI, and many will provide a GUI as well.

**User Application Layer**

One of the primary objectives of MONTE, as described above, is to deliver software that is easy to use. This means different things to different people, and can depend on a number of factors. Novice users of an application often find that a GUI is easier to learn to use than a CLUI. The GUI presents the features of the software to the user, making it possible for the user to discover the steps needed to solve a particular problem. Although a well-designed GUI will not unnecessarily impede an expert user, it will usually provide little benefit. In a CLUI-based application, on the other hand, individual commands must be memorized or found in a User Manual. This may be exactly what the expert user desires, but often proves to be frustrating for a novice.

Clearly, neither of these solutions is appropriate in all cases. Ultimately, it is reasonable to expect that the majority of MONTE users will fall into the expert category, but there will always be new users that start out as novices. MONTE must strike a balance between ease of use for the novice and for the expert. This will be accomplished by providing a variety of end-user interaction modes. Many applications will be delivered with both graphical and command-line interfaces.

In addition, MONTE will provide a scripting capability based on the Python programming language that will allow users to quickly develop prototype solutions to new problems by building on the functionality of the MONTE Toolkit Layer. This scripting capability will be composed of stand-alone scripts that can
be run from a Unix prompt, written either by the user or by the MONTE team, and an interactive command-line environment that will provide access to the underlying functionality of the Toolkit Layer. Python is an interpreted language, like Perl, and hence is ideal for rapid prototyping. It is, however, a rigorously designed, full featured, object-oriented language that is easier to learn and understand. Python also has several useful language extensions, including the NumPy numerical extensions (for MATLAB-like functionality), Tk bindings (similar to TCL-Tk), and PyQt bindings for the Qt GUI toolkit. Python has also been designed to integrate easily with C/C++, which makes it possible to provide access to essentially any part of MONTE through a Python interface.

CONCLUSIONS

In summary, MONTE is a new software system currently being developed to solve a variety of problems in the areas of navigation and trajectory design of space missions. MONTE will extend the functionality of the legacy software, reduce long-term maintenance costs, and provide improved usability to the mission and navigation analysts. These objectives will be accomplished through rigorous analysis of the system requirements, intelligent application of object-oriented technology, a flexible and robust software architecture, and appropriate attention to formal software development process issues.

Fig. 1. Logical Structure of the MONTE System
ACKNOWLEDGMENTS

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Copyright 2012 California Institute of Technology. Government sponsorship acknowledged.