Using Mach Threads to Control DSN Operational Sequences

Juan Urista

Monitor & Control Technology Group
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA, 91109-8099, USA
MS 525-3660

ABSTRACT

The Link Monitor and Control Operator Assistant prototype (LMCOA) is a state-of-the-art, semi-automated monitor and control system based on an object-oriented design. The purpose of the LMCOA prototyping effort is to both investigate new technology (such as Artificial Intelligence) to support automation and to evaluate advances in information systems toward developing systems that take advantage of the technology (Ref. 1).

The emergence of object-oriented design methodology has enabled a major change in how software is designed and developed. This paper describes how the object-oriented approach was used to design and implement the LMCOA and the results of operational testing. The LMCOA is implemented on a NeXT workstation using the Mach operating system and the Objective-C programming language.

Key Words: Mach, thread, monitor and control, operator assistant, object-oriented programming, Deep Space Network.

1.0 Background on LMCOA Prototype and Functional Description

The Link Monitor and Control Operator Assistant (LMCOA) prototype demonstrates semi-automated monitor and control for a Deep Space Network (DSN) station. The LMCOA consists of four major modules: a TDN Execution Manager, a Situation Manager, a Router and a Monitor Data Base. The LMCOA also contains multiple, concurrently executing TDN Block modules. The TDN Execution Manager is responsible for traversing the TDN and starting execution of each TDN Block at the appropriate time. Each TDN Block verifies preconditions and postconditions with the Situation Manager and sends its internal list of directives sequentially. The Situation Manager maintains a device model to track the predicted and actual state of the system. It provides closed loop control, anomaly detection and recovery assistance. The Router is responsible for traffic between the LMCOA and the other modules. The Monitor Data Base stores subsystem data for future querying by the Situation Manager.

The LMCOA prototype uses a temporal dependency network (TDN) to represent LMC operations procedures. A TDN is a directed graph that incorporates temporal and behavioral knowledge and provides optional and conditional paths through the network. The directed graph (or, Petri Network) represents the steps required to perform an operation. Precedence relationships (step A has to happen before step B) are specified by the nodes and arcs of the network. The behavioral knowledge identifies system-state dependencies in the form of pre- and post-conditions. Temporal knowledge consists of both absolute (e.g., Acquire the spacecraft at time 02:30:45) and relative (e.g., Perform step Y 5 minutes after step X) temporal constraints. Conditional branches in the network are those performed only under certain conditions. These are the IF (this condition) THEN (do/don't do that action). The conditionals are used primarily for error recovery. Optional paths are those which are not essential to the operation -- but may, for example, provide a higher level of confidence in the data if performed (Ref. 2).

2.0 LMCOA Object-Oriented Design

Object-oriented design is a methodology for managing data and the functions that act upon the data. The data and its functions are known as a complex object. Major principles for decomposing complex objects into a manageable form include classification, encapsulation and inheritance. Classification is a mechanism for identifying similar things. For example, in the LMCOA, individual directives can be classified into a general directive class because all directives share basic traits such as a destination subsystem, a source subsystem and a command with parameters. Even though the value for these elements may vary for specific directives, the attributes of a directive remain the same. The specific elements of a class, where the attributes are filled with some value, are called objects of a class.

Encapsulation is a principle for combining data and specific methods that act on the data into one entity -- a class. A user of a class sees only the methods for interaction -- the detailed data remains hidden. This results in simple and modular interfaces to system modules that behave in very specific ways. Furthermore, encapsulation eases the task of code maintenance and upgrading. The internals may be upgraded and as long as the external interface remains the same -- the users of the object will never need to know about the upgrade. The upgrade is virtually invisible to the users of the object (Ref. 6,7).

Inheritance is another feature of an object oriented development environment. It is a mechanism for expressing similarities among classes. For example, in the LMCOA, there are 'control directives' and 'display control directives.' Control directives are issued to the antenna and associated subsystems. Display control directives are issued to the monitor and control system in order to bring up displays. In the LMCOA, we define a general class called 'directives' and then create a subclass called 'control directives' which inherits most of its behavior from the more general 'directives' class.

2.2 Mapping of LMCOA Architecture Into Objects

A TDN is a complex object that encodes the information necessary to perform a specific operational task. As described earlier, the primary representation of the TDN is an augmented directed graph. In the graph, each arc represents a strict precedence relationship, each node a sequence of directives that perform a subset of the overall function. The network explicitly specifies the precedence relationships between nodes, any potential parallelism, and rules for recovering from global faults. The nodes, or blocks, consist of the directives, temporal constraints, pre- and post- conditions, and local recovery information should the block fail.

The TDN block object contains the methods that read the directives and check the block pre- and post- conditions. These conditions notify the block whether the directive has been satisfied. The block is activated by the TDN class and provides a mechanism to allow concurrent processing. Each active block is a thread that is interleaved by the operating system kernel. The blocks cycle through its execution reading directives and checking its pre- and post-conditions. Once the block has completed, conditions are established for the next block to be executed. These conditions notify the TDN that activates the next block. The Directive object contains the directive string obtained from the DSN. This instance contains the data necessary to format and create the DSN block. The DSN block, known as the DSN Standard Subsystem Block or SSB, is the data format required for DSN subsystem communications (Ref. 4).

The Router object contains the methods that provide the communication path between the DSN and the LMCOA. The Router consists of a socket connector and a listener. The socket connector uses UNIX 4.3 BSD sockets to establish communications to the DSN network. The socket is connected to a communications program residing on an IBM compatible Personal Computer (PC 286AT) installed with an IEEE communications board that is required for communications with the DSN network. The communications program reads data from the DSN interface network unit and routes the data through a PC ethernet socket. The listener is

spawned as a thread and checks the socket port waiting for data to arrive.

The Situation Manager object is responsible for updating the action model. The updates are based on the directives being sent and the responses and event notification messages being received (Ref. 3).

3.0 Implementation of Object-Oriented Design

Each major module in the design is represented as an object. For example, the LMCOA defines the objects Directive, Block, and TDN. Each object contains the necessary data and behavior actions used by the LMCOA. Every time a directive is issued, it is an instance of the directive object. The block object is responsible for issuing the directive, while the TDN manager controls which blocks are active. The major modules of the LMCOA are shown in Figure 1.0. The modules correspond to objects used by the LMCOA.

The execution of the LMCOA begins with the TDN object. The TDN object identifies which block object is ready to begin execution. Once active, the block sequetially sends its list of directives to the DSN subsystems. The block is active until all the block's postconditions are satisfied. This is achieved when the DSN subsystems respond to the directives, issue data to the LMCOA through directive responses and/or event notice messages, and the Situation Manager verifies that the directives executed correctly. The responses and event notification messages are obtained through the Router object, which sends the data to both the block objects and the Situation Manager object.

3.1 General Description of Threads

A thread is an operating system construct which is the basic unit of process execution and scheduling. Each thread carries only the portion of the processor state that is necessary for independent execution. Thus threads must reside within a normal UNIX process which carries the entire processor state. However, multiple threads can execute concurrently within the context of a single process. Multiple threads, executing within a process, share the same memory space. A change in shared data by one thread can be seen by all other threads in the process (Ref. 5). In a multi-processor environment, concurrently executing threads on separate processors can vastly improve execution time. In a single processor environment, threads are extremely useful for logical concurrency such as when several actions need to be taken on the data at the same time. For example, the major LMCOA modules are implemented as threads for simultaneous execution on the same data from the DSN subsystems. In the LMCOA, threads are also useful for interleaving directive execution and as a way to default scheduling of concurrent components to the operating system.

3.2 How Threads Were Used in LMCOA

Before the TDN Execution Manager begins the execution cycle, the LMCOA loads the TDN files and spawns the Situation Manager and the Router. The Situation Manager thread executes and begins by loading the knowledge base. After which, the Situation Manager waits for input from other modules. DSN monitor data blocks are routed to the LMCOA monitor database for later retrieval by the Situation Manager.

The LMCOA design uses multiple threads to achieve the scheduling and execution of directives. In the LMCOA, the directives are grouped according to functions known as TDN Blocks. The LMCOA begins execution by spawning the TDN object as a thread. This thread is the execution manager of the LMCOA. The operator starts the TDN execution by selecting the start button that invokes the "startTDN" method. This method invokes and spawns the start TDN Block object as a thread. The TDN start block starts the TDN blocks as parallel executing processes. The parallel block objects are spawned as threads and the operating system kernel time-slices each thread. Time-slicing the threads allows the LMCOA to execute each of the threads as interleaved processes. The LMCOA defaults the execution of the threads by allowing the kernel to schedule the threads in a logical and fair concurrent order.

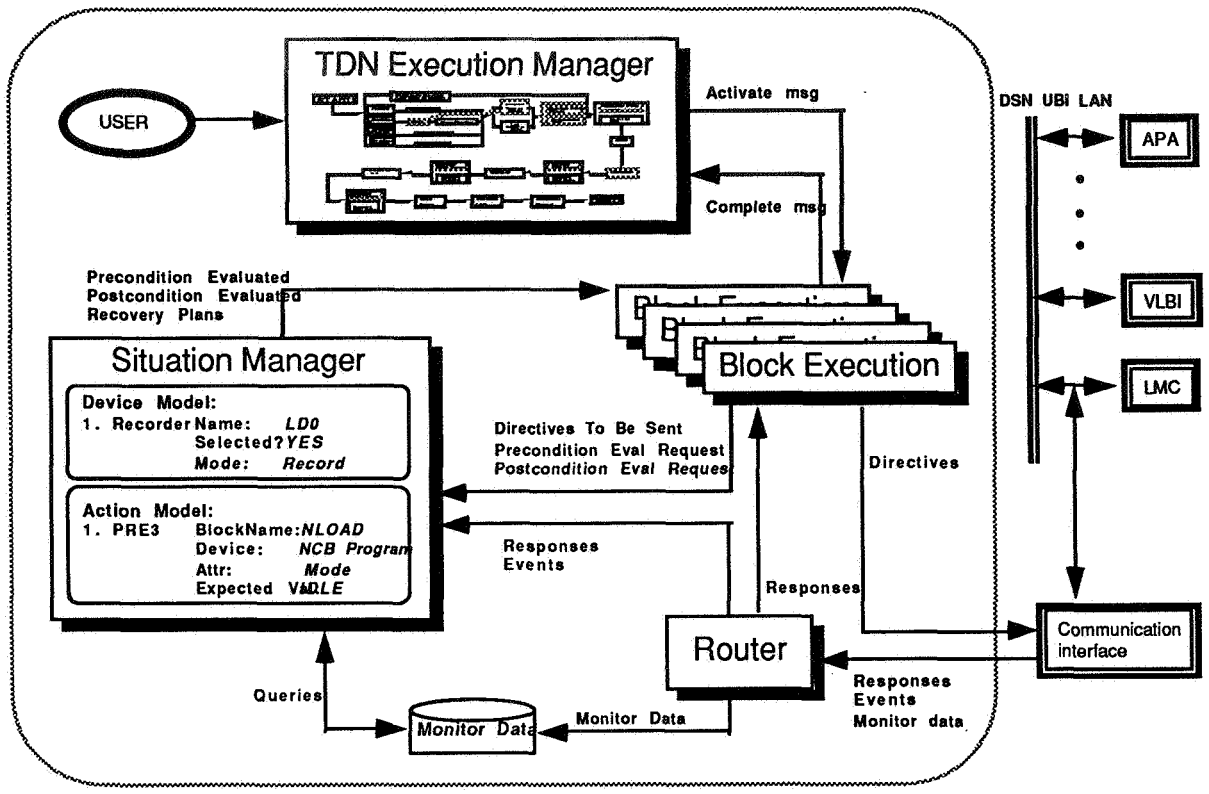

Figure 1.0 LMCOA Major Modules and Control Flow

The blocks start the process of sending directives by invoking the directive object. The directive object encapsulated methods are used by the block thread. A block can have a number of directives executing or waiting to be executed. When the block is ready, it sends the directive to the DSN subsystem. Figure 2.0 illustrates some of the threads that are spawned by the LMCOA.

Each of these threads remains within the same memory space of the LMCOA process. The threads are released when they are done executing. As shown in Figure 2.0 the TDN Execution Manager thread spawns block threads "Load APA Predicts" and "APA Precal" in parallel with block threads "Load VLBI Predicts", "VLBI Precal", and ""PPM Precal." These block threads are implemented in the LMCOA as a precalibration directive sequences for the DSN subsystems Antenna Pointing Assembly (APA), Very Long Baseline Interferometry (VLBI), and Precision Power Monitor (PPM).

3.3 Example

To illustrate how threads operate in the LMCOA, Figure 3.0 shows a portion of the LMCOA Temporal Dependency Network. Table 1.0 shows a typical life cycle of the block threads shown in Figure 3.0. The TDN, Router, and Situation Manager threads span the entire program life cycle of the LMCOA process. Blocks, for example block "Load APA Predicts" execute only during a portion of the time. This block thread formats and sends directives to the DSN Antenna Pointing Assembly subsystem. Concurrently, the "Load VLBI Predicts" block thread is executed which, in turn, formats and sends the directive to the DSN Very Long Baseline Interferometry subsystem.

These threads execute in parallel and are able to share the same memory space. Thus allowing for an efficient use of both data sharing and module execution.

4.0 Status and Results

In September of 1992, the object oriented implementation of the LMCOA was tested at the Goldstone Deep Space Communications Complex. The LMCOA operated the 70-meter antenna in parallel with the existing LMC subsystem. The LMCOA object oriented design and use of the Mach operating system allowed the operator to issue parallel directives to the Antenna Subsystem and the Very Long Baseline Interferometry Subsystem. The LMCOA successfully issued the directives and processed the directive responses and subsystem event notification messages. This test allowed us to evaluate the design and Mach operating system and the results indicated that the use of objects and threads lends itself well into the designing systems that require concurrent and parallel execution. Additional testing is continuing on the knowledge base and the Situation Manager.

5.0 Acknowledgments

The work described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. The other members of the LMCOA team whose contributions help support this work are Sanguan Chow, Kristina Fayyad, Randall W. Hill Jr, and Kathy Sturdevant. Special thanks to Lynne P. Cooper and Lorrine Lee whose encouragement, patience, and enthusiastic support made all this work possible.

6.0 References

1. Cooper, Lynne P., Rajiv Desai and Elmain Martinez, "Operator Assistant to Support Deep Space Network Link Monitor and Control." SOAR Symposium, Houston, TX. 1991.

2. Fayyad, Kristina E., and Lynne P. Cooper, "Representing Operations Procedures Using Temperal Dependency Networks", SpaceOPS 92: Second International Symposium on Ground Data Systems for Space Mission Operations, Pasadena, CA., 1992.

3. Lee, Lorrine and Randall W. Hill, Jr., "Process Control and Recovery in the Link Monitor and Control Operator Assistant," SOAR Symposium, Houston, TX. 1992.

4. JPL Internal Document 890-131, Deep Space Network DSCC General Data Flow Standards, Version 1.0, Revision C., November 14, 1990.

5. NeXT Operating System Software, NeXT Computer, Inc, 1990.

6. Coad, P., and E. Yourdon, *Object-Oriented Analysis*. Yourdon Press, 2nd Edition, 1991.

7. Boach, G., *Object-Oriented Design With Applications*. Benjamin Cummings, 1991.

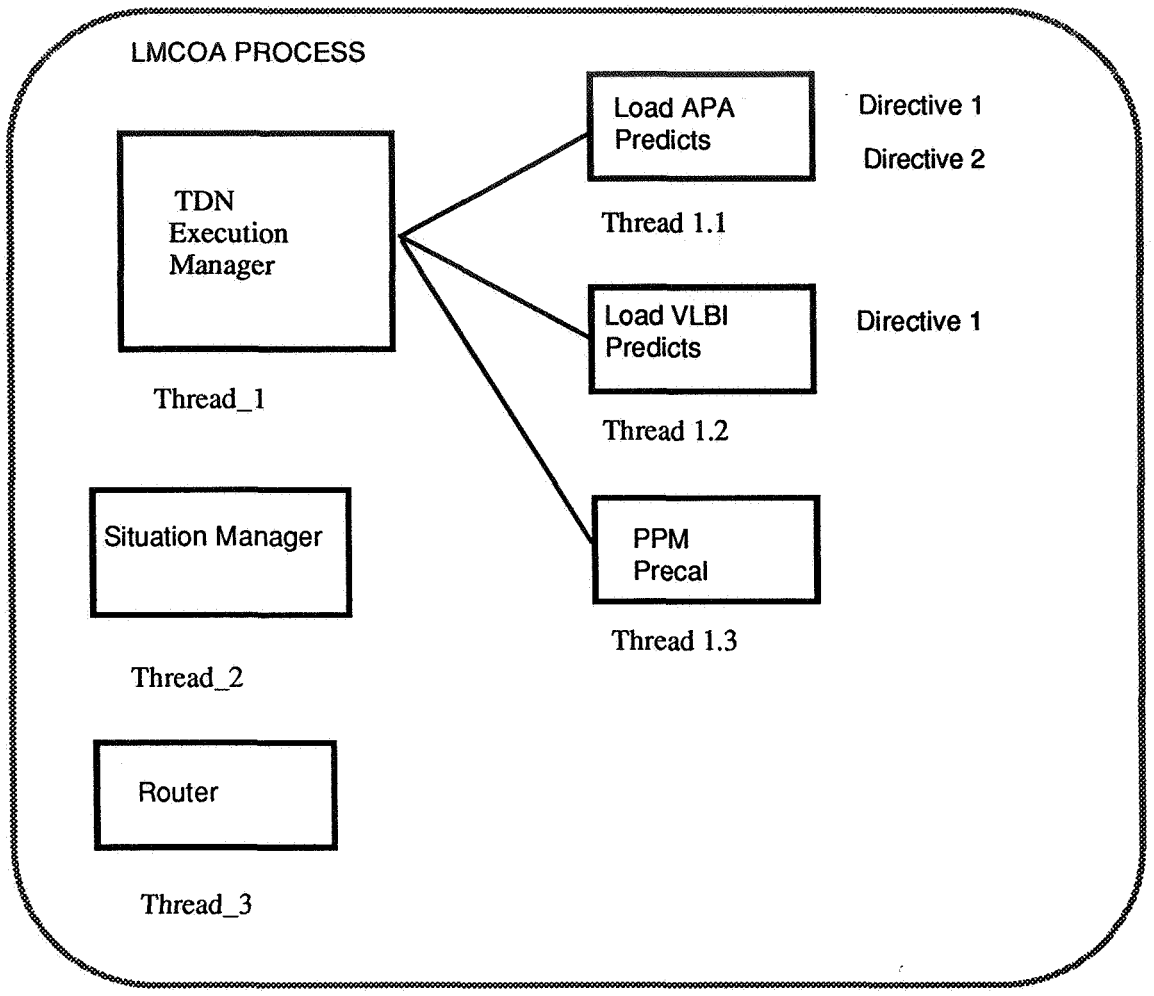

Figure 2.0 LMCOA Object as Threads

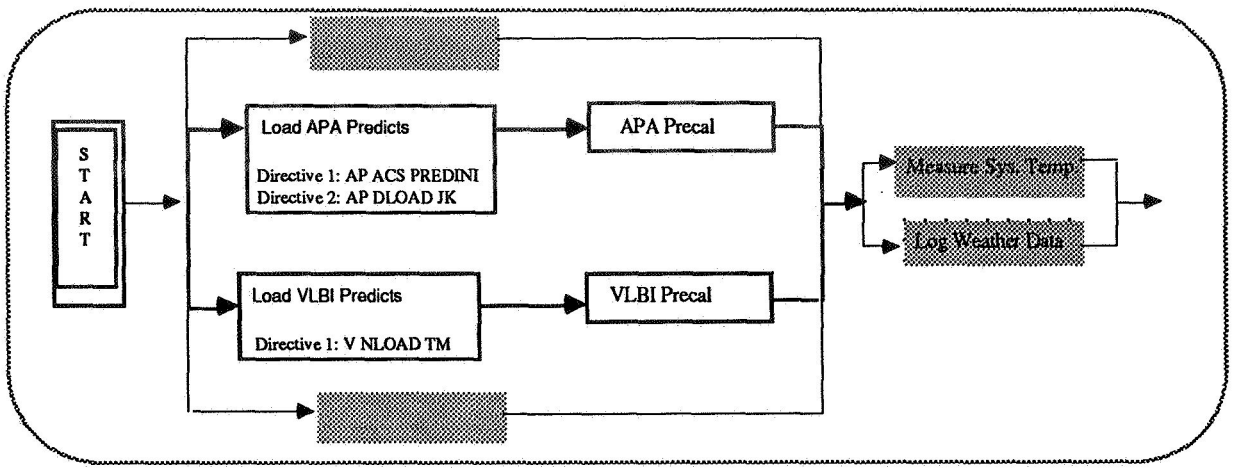

Figure 3.0 LMCOA Partial Temporal Dependency Network Representation

Object	Time --- LMCOA Process Execution Cycle
TDN	
Router	
Situation Manager	
"Load APA Predicts"	
Directive 1: AP ACS PREDINI	
Directive 2: AP DLOAD PRED JK	
"Load VLBI Predicts"	
Directive 1: V NLOAD TM	
"APA Precal"	

Table 1.0 LMCOA Process and Thread Life Cycles