# A PROBABILISTIC MODEL OF THE EFFECTS OF SATELLITE SYSTEM AUTOMATION ON COST AND AVAILABILITY

by

**Robert E. Schwarz**

B.S. Mechanical Engineering Aerospace Option
Rutgers College of Engineering, 1995

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MAY 1997



Signature of Author: _____
Department of Aeronautics and Astronautics
May 9, 1997

Certified by: _____
James K. Kuchar
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by: _____
Prof. Jaime Peraire
Chairman, Committee for Graduate Students

# A PROBABILISTIC MODEL OF THE EFFECTS OF SATELLITE SYSTEM AUTOMATION ON LIFE CYCLE COSTS AND SYSTEM AVAILABILITY

by

**Robert E. Schwarz**

Submitted to the Department of Aeronautics and Astronautics
on May 9, 1997 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Aeronautics and Astronautics

## ABSTRACT

A probabilistic methodology to examine the impact of satellite system task automation on cost and availability is described. The methodology uses a functional representation of the system and discrete levels of automation are defined for the human to ground computer and ground station to satellite interfaces for each function. The level of automation in turn affects development and operations costs, the likelihood of successfully managing an event, and the processing time required. A Markov model is used to determine the resulting availability of each function and ultimately the ability of the satellite system to carry out its mission objectives. The methodology has been implemented in an interactive software tool called SOCRATES which facilitates the construction of satellite system architectures and the analysis of different operational concepts. SOCRATES was then used to investigate automation trades for three case studies. The first study determined the effects of automating the tracking function of a single satellite through the use of an on-board GPS receiver. For the inputs used, it was shown that full automation of the tracking function resulted in the minimum life cycle costs for the system. A second case study illustrated the automation trade for a constellation. The payload function was automated, and for the model inputs used, it was shown that a low level of automation was optimal for small constellations, while high automation was optimal for large constellations due to economies of scale. Automation was applied to a deep space satellite in the third case study. It was shown that the delay time for distant missions made it necessary to automate functions which require frequent interaction.

Thesis Supervisor: James K. Kuchar

Title: Assistant Professor of Aeronautics and Astronautics

# ACKNOWLEDGMENTS

I would like to first thank my wife, Sandy for her support over the last few years. I know that I would not be where I am without her help, love, and encouragement. I would also like to thank my son, Ryan for reminding me to play every now and then. I thank my parents for their financial and unconditional moral support, as well as their willingness to let me learn from my own mistakes.

Much thanks are due to my thesis advisor, Prof. James Kuchar, whose suggestions and support over the last two years has been invaluable. I am very grateful for his dedication to the quality of my work, and for the long hours he spent editing papers and presentations, as well as this thesis. Thanks are also due to Prof. Daniel Hastings, who made many contributions to the research and helped me understand the value of strong communications skills. I thank Stephan Kolitz for helping me to maintain a consistent focus on my research, and for making valuable suggestions along the way. I would like to thank Prof. John Deyst for his modeling suggestions early in the research which helped form the Markov modeling foundation upon which this work is based. I would also like to thank Aidan Low for his help with the SOCRATES graphical user interface, and to Sabrina Almeida for her work with spacecraft failure statistical analysis. I would like to thank the members of the Aeronautical Systems Lab and the Space Systems Lab at MIT for their friendship as well as their technical advice and support. I also thank the Charles Stark Draper Laboratory for providing funding for this project.

# Table of Contents

# TABLE OF FIGURES

# TABLE OF TABLES

# Acronyms, Abbreviations, and Nomenclature

| | |
|---|---|
| **$\alpha(A(t))$** | Elasticity factor as a function of availability |
| **A(MO)** | Mission objective availability |
| **A(O)** | Objective availability (satellite objective or control center objective) |
| **A(RS)** | Redundancy string availability |
| **A(t)** | Availability as a function of time |
| **ACS** | Attitude Control System |
| **$A_d$** | Availability accounting for dependencies |
| **$A_i$** | Availability assuming independence |
| **AR(t)** | Actual revenue as a function of time and availability |
| **$A_{ss}(func_i)$** | Steady state availability of function i |
| **BOL** | Beginning of Life |
| **$C_{FU}$** | Theoretical first unit cost |
| **C.O.** | Control Center Objective |
| **DC** | Development costs |
| **dt** | Discrete time Markov model time step |
| **EOL** | End of Life |
| **ER(t)** | Expected revenue as a function of time |
| **FDIR** | Fault Detection Isolation and Recovery |
| **GEO** | Geostationary orbit |
| **GPS** | Global Positioning System |
| **GUI** | Graphical User Interface |
| **H** | The probability that a human is involved in the recovery process |
| **LCC** | Life cycle costs |
| **LEO** | Low Earth Orbit |
| **LT** | Logged time - the amount of time a human actually spends on recovery |
| **M(t)** | Market factor as a function of time |
| **MANS** | Microcosm Autonomous Navigation System |

| | |
|---|---|
| **MIT** | Massachusetts Institute of Technology |
| **mmtf** | Failure mode's mean time to failure |
| **M.O.** | Mission Objective |
| **mte** | Mean time to event |
| **mtf** | Mean time to failure |
| **mtpf** | Mean time to permanent failure |
| **mtr** | Mean time to recovery |
| **mttc** | Mean time to completion |
| **$mtt_g$** | Ground processor transition time |
| **$mtt_s$** | Space processor transition time |
| **mttx** | Mean time to transition |
| **$N_E$** | Total number of operators for event recovery |
| **$N_{E2}$** | Number of operators for event recovery due to type 2 dependencies |
| **$N_{E3}$** | Number of operators for event recovery due to type 3 dependencies |
| **$N_{EI}$** | Number of operators required for independently occurring events |
| **$N_F$** | Number of operators required for failure recovery (all modes) |
| **$N_{Fm}$** | Number of operators required for a failure mode recovery |
| **$N_S$** | Number of operators for supervisory tasks |
| **O** | Objective (satellite or control center) |
| **OBP** | On Board Processor |
| **OC** | Operations costs |
| **OPC** | Opportunity costs |
| **OT** | Overall time - the total elapsed time for human recovery |
| **P(perm.fail)** | Probability a permanent failure has occurred |
| **P(success)** | Success probability |
| **P(tx)** | Transition probability |
| **PC** | Personal Computer |
| **pdf** | Probability Density Function |
| **$P_G$** | Ground processor success probability |
| **$P_S$** | Space processor success probability |

| | |
|---|---|
| **PWV** | Present Worth Value |
| **R** | Revenues |
| **$R_{max}$** | Maximum possible revenue |
| **RS** | Redundancy string |
| **S** | Learning curve (%) |
| **$S_E$** | Salary for operators performing event recovery |
| **$S_F$** | Salary for operators performing failure recovery |
| **$S_S$** | Salary for operators performing supervisory tasks |
| **S.O.** | Satellite Objective |
| **SOCRATES** | Satellite Operations Cost and Reliability Analysis Toolkit for the Evaluation of Systems |
| **t** | Time |
| **TPC** | Total Production Costs |
| **USCM7** | Unmanned Spacecraft Cost Model, Version 7 |

# 1. Introduction

## 1.1 Background

Current levels of automation in satellite systems reflect an incremental evolution that is based on a high level of human involvement. Historically, this has been a result of the desire to reduce risk and due to limited technological capabilities. Due to the dependence on humans to perform tasks, operations costs can make up a significant portion of the life cycle costs for a satellite system.[1] In addition, human error continues to be a major cause of spacecraft anomalies and failures.[1] With the introduction of large constellations or clusters of satellites, some automation of operations will be required to reduce costs while maintaining availability (the probability of meeting system requirements at a given time).[2-5]

Despite the recognition of the need for automation, there is reluctance to do so. The large investments and high risks involved in space ventures has lead to a conservative industry. In addition, the desire to reduce cycle times for new programs also favors significant re-use of proven technologies and thus, low levels of automation. A methodology which can be used to predict the effects of automation on a satellite system would help reduce risk and may enable more cost efficient systems to be considered. The definition of performance metrics, such as the system availability (probability of meeting system requirements) and life cycle costs (sum of development, operations, and opportunity costs) would enable operations concepts to be quantitatively compared.

Figure 1.1 illustrates a typical satellite system and communications links among the system's decision makers (shown in gray). The system consists of a space segment (satellites) as well as a ground segment (control centers) which in turn consist of computers and human operators. The satellite contains sensors which measure the status of the satellite and pass that data to an on-board processor (OBP). The OBP may interpret the data and generate commands to control actuators used to keep the satellite operating as intended. Or, the data may be sent to the ground computer through a communications link. The ground computer, using the downlinked data and/or information from ground-based sources such as radar may then command the satellite actuators through the link to the OBP. Additionally, the data may be sent to an operator through a human machine interface. The human operator must then determine when and what command action is to be taken. This may or may not also require interaction with additional engineering support staff. The operator then commands the satellite through the ground computer and OBP. Thus, three types of processors can be involved in the decision making process: the OBP, ground computers, or the human operator.

The measure of automation in a system depends on the degree to which each processor (OBP, ground processor, human) is involved with controlling the spacecraft. This level of automation can vary along a continuum ranging from fully-manual human control, to human supervisory control, to a fully automated (no human) system.[6] In general, as the level of automation is increased, fewer tasks need to be performed by the human operator and operational costs decrease.[4] In addition, when properly applied, automation can result in a reduction in the rate of human-caused errors (e.g., miss-typed commands). However, because an automated system may not be as flexible as a human operator in managing unanticipated situations, availability could decrease. Finally, the increased development costs associated with a highly-automated system may outweigh any operational cost benefits. Thus, the appropriate degree of automation requires a careful trade study between three factors: operating costs (personnel), development costs (design and infrastructure), and availability.

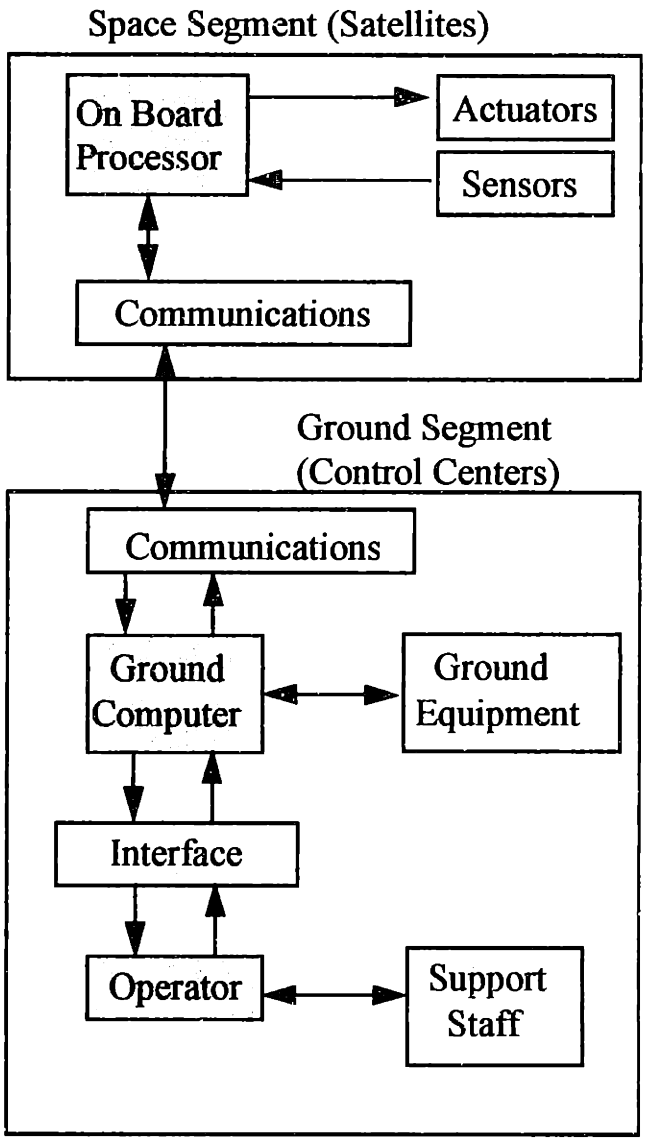


*Figure 1.1: Block Diagram of a Typical Satellite System*

There is also a trade concerning the degree to which automation is resident on the OBP vs. the ground computer. The OBP has the advantage of being on the spacecraft which eliminates communications problems with sensors and actuators. However, the OBP must be specially designed to withstand the space environment. As a result, spacecraft processors tend to be several years behind terrestrial technology. This limits the OBP in both memory and throughput relative to ground computers. Also, the ground computer may be replaced if it fails or be upgraded to become more effective. This makes ground computers more flexible and reliable.

## 1.2 Notional Effects of Automation

Figure 1.2 qualitatively represents the cost and availability characteristics of a hypothetical satellite system with respect to an increasing level of automation. As low levels of automation are introduced into the system, the operating costs decrease, principally due to a decrease in the number of human operators (Figure 1.2A). At some point, however, the increase in design and development costs (due to software development) may outweigh the decrease in operation costs.

As shown in Figure 1.2B, availability may decrease, increase, or be unaffected by the level of automation. For tasks that are simple, well understood, or periodic (such as routine stationkeeping on a geostationary satellite), availability may increase with increasing automation (Task A in Figure 1.2B). This trend results when human errors are more likely than software errors and the impact of unanticipated situations is negligible. For complex, rare, or unexpected functions, availability may decrease as humans are removed from the loop (Task C in Figure 1.2B). This can occur when the automation is unable to resolve problems that could have been resolved by a human or when the automation fails to accurately inform the human of the situation. Alternatively, there may be some functions for which the availability is nearly independent of the level of automation (Task B).

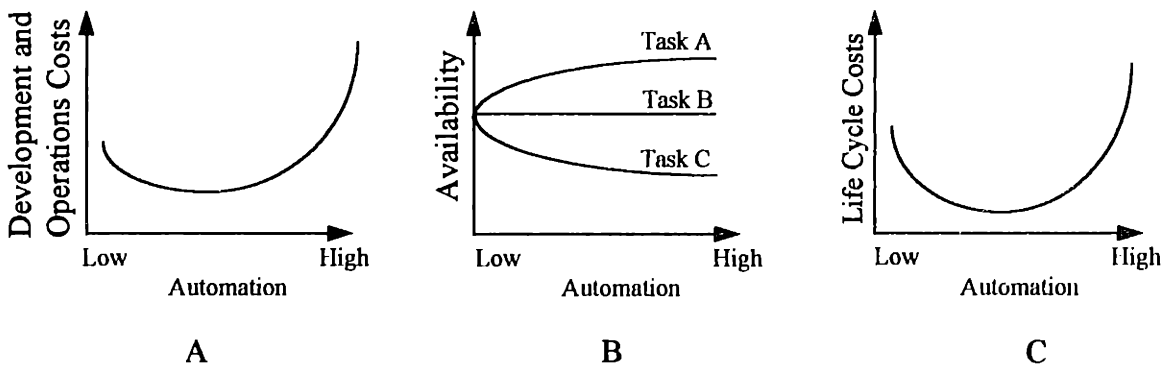


*Figure 1.2: Effect of Automation on Cost and Availability*

An increase in system availability translates into an increase in revenues for a commercial system, or an increased ability to perform an objective for science or military systems. Thus, the potential for failure can be represented by an opportunity cost which represents revenues forgone as a result of increased system down-time. For science or military applications, the definition of an opportunity cost may be difficult. In such cases, the development and operations cost would be compared against the availability metric without attempting to define the life cycle cost. However, for commercial systems, the opportunity cost can be added to the development and operations costs to form the life cycle cost metric. Determination of opportunity costs requires additional data such as the relationship between a particular function and revenue. The combination of the two curves from Figure 1.2A and Figure 1.2B are represented in Figure 1.2C, showing the overall life cycle cost. In its simplest form, the life cycle cost metric, LCC, is therefore calculated as:

$$LCC = DC + OC + OPC(R, A) \tag{1.1}$$

where DC is the development cost, OC is the operations cost, and OPC(R, A) is the opportunity cost as a function of revenue and availability.

In the example in Figure 1.2, there exists an optimal level of automation at which life cycle cost is minimized. Due to the complexity of the satellite system, a methodology is needed that can model the effect that automation has on costs and system availability. Such tools would enable system engineers to identify those functions that should be automated. The primary focus of the methodology presented here is to develop a model of a satellite system to aid in quantifying the curves in Figure 1.2.

## 1.3 Roadmap

This thesis presents a methodology by which the cost and availability effects of satellite automation can be modeled. A discussion of the general concepts and an outline of the approach is contained in Chapter 2. The methodology has been implemented in a detailed

model. The availability portion of this model is described in Chapter 3, and the cost portion in Chapter 4. The software tool, SOCRATES, which is based upon the model is outlined in Chapter 5. Examples of how the model can be used in systems studies are shown using case studies in Chapter 6. Chapter 7 contains a summary of the work and points out the capabilities and limitations of the model.

# 2. Methodology

## 2.1 Overview

While Figure 1.1 illustrates a block diagram of the physical system, one may also think of the system in a more abstract manner. One such representation of the system is based upon the functions it performs, as shown in Figure 2.1. These functions represent actions which make up the operations of a space system. Automation is introduced to the system by automating, to varying degrees, the separate functions of the system. Typical functions include stationkeeping, attitude control, payload operation, power, and so on. Ground functions such as archiving or command generation are also included since automation may be applied to these tasks as well as to satellite functions. At any given time, there is some probability that a function is fully operational (not degraded). This operational probability is defined as the function's availability at that time.

## 2.2 Mission Objectives

The overall system has been designed to perform one or more *mission objectives*. These mission objectives are top level goals which provide some value to the system's users. For commercial systems, such as a TV broadcasting system, there may only be one mission objective: to provide a TV signal to the intended areas on the surface of the Earth. A scientific satellite may have multiple mission objectives, each of which may result from the data received by multiple payload sensors.

Each satellite or control center in the system may have objectives of its own, namely, *satellite objectives* (S.O.) and *control center objectives* (C.O.). These objectives are the goals which the individual satellites or control centers must achieve. Mission objectives are met through a combination of satellite and control center objectives. For example, the science system may have a mission objective which is to record and store infra-red signatures of the Pacific Ocean. This would be accomplished through a combination of satellite objectives, such as the infra-red data collection and orbit contr', as well as control center objectives such as data archiving. Mission objectives require certain satellite and control center objectives to be functional. If any of these lower level objectives are not available at a given time, the mission objective is not met.

Satellite and control center objectives are achieved through the action of space and ground functions as shown in Figure 2.1. For example, the data archiving control center objective may require the telemetry function on the satellite to be operational. Thus, a space function is required to perform a control center objective. Functions are elemental tasks which the system must perform to achieve lower level objectives, and hence mission objectives.

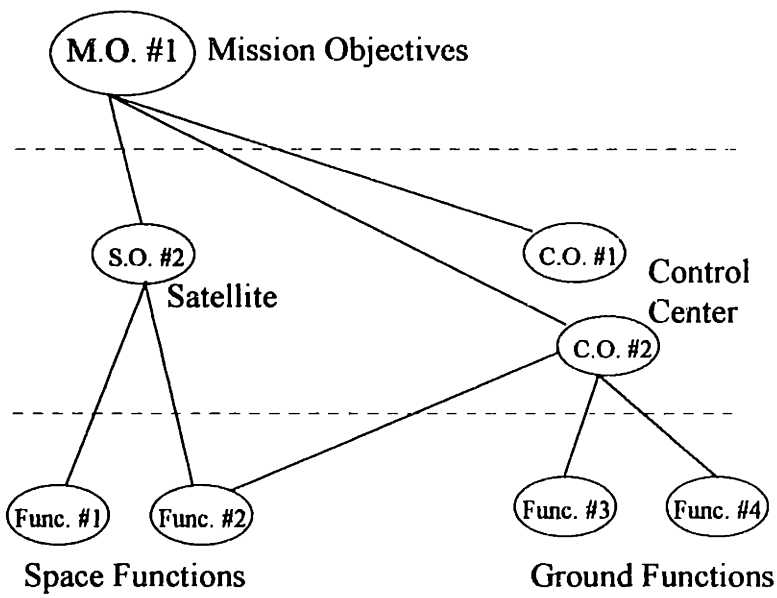


*Figure 2.1: Example Functional Representation of System*

The level of detail of the model is reflected by the number of functions which are defined. For example, one approach may be to define attitude control as a function of the system. Another might further decompose this function by defining roll angle control, yaw angle control, and pitch angle control as separate functions of the system. Neither approach is more correct, but one may be more appropriate for certain applications of the methodology.

## 2.3 Approach

The main goal of the methodology is to determine the effects that automation has on the life cycle cost of the entire system. As was indicated in Figure 1.2, automation is expected to impact development costs, operations costs, and the system's availability. The system's availability is measured by the system's mission objective availabilities. In order to model these impacts, sub-models for the availability, operations costs, and development costs are needed, as shown in Figure 2.2.

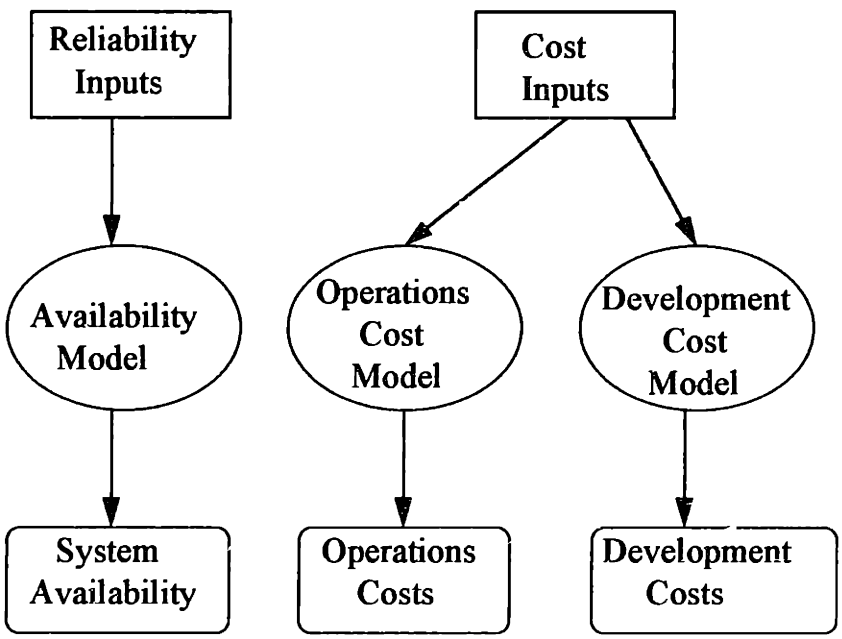


*Figure 2.2: Sub-Model Components*

Reliability inputs such as human and computer processor task reliabilities, hardware reliabilities, and task execution times, are used by the availability model to calculate the availability of each function. These are used, along with satellite, control center, and mission objective requirements, to calculate each mission objective's availability. The details of this model are discussed in Chapter 3. Cost inputs such as production, launch, administrative costs, and operator salaries are used by the development and operations cost models to calculate the development and operations costs. The details of the cost models are included in Chapter 4.

The operations costs are affected by the number of operators required to perform each function. This, in turn, is affected by the workload imposed by failures and other operations, which is captured in the system availability model. Therefore, in addition to cost inputs, the operations model also requires an estimate of the system's availability in order to determine the staffing requirements (Figure 2.3).

The availability can also be converted into an opportunity cost representing forgone revenues. This is then combined with the development and operations costs to derive a life cycle cost metric. The life cycle cost was defined by Eq. 1.1, restated below, and is equal to the sum of the present worth values of development, operations, and opportunity costs. The opportunity costs are a function of the potential mission objective revenues and their availabilities. This relationship may also include the impact of system availability on the consumers' demand on the service provided, as well as expected variations in the demand for the service as a function of time. Therefore, a market model (as shown in Figure 2.3) is required to calculate the opportunity cost function OPC(R,A).

$$LCC = DC + OC + OPC(R, A) \tag{1.1}$$

An integrated cost model is required to convert availability to opportunity cost, and to combine the costs as the present worth value of the life cycle costs. This is shown in Figure 2.3.

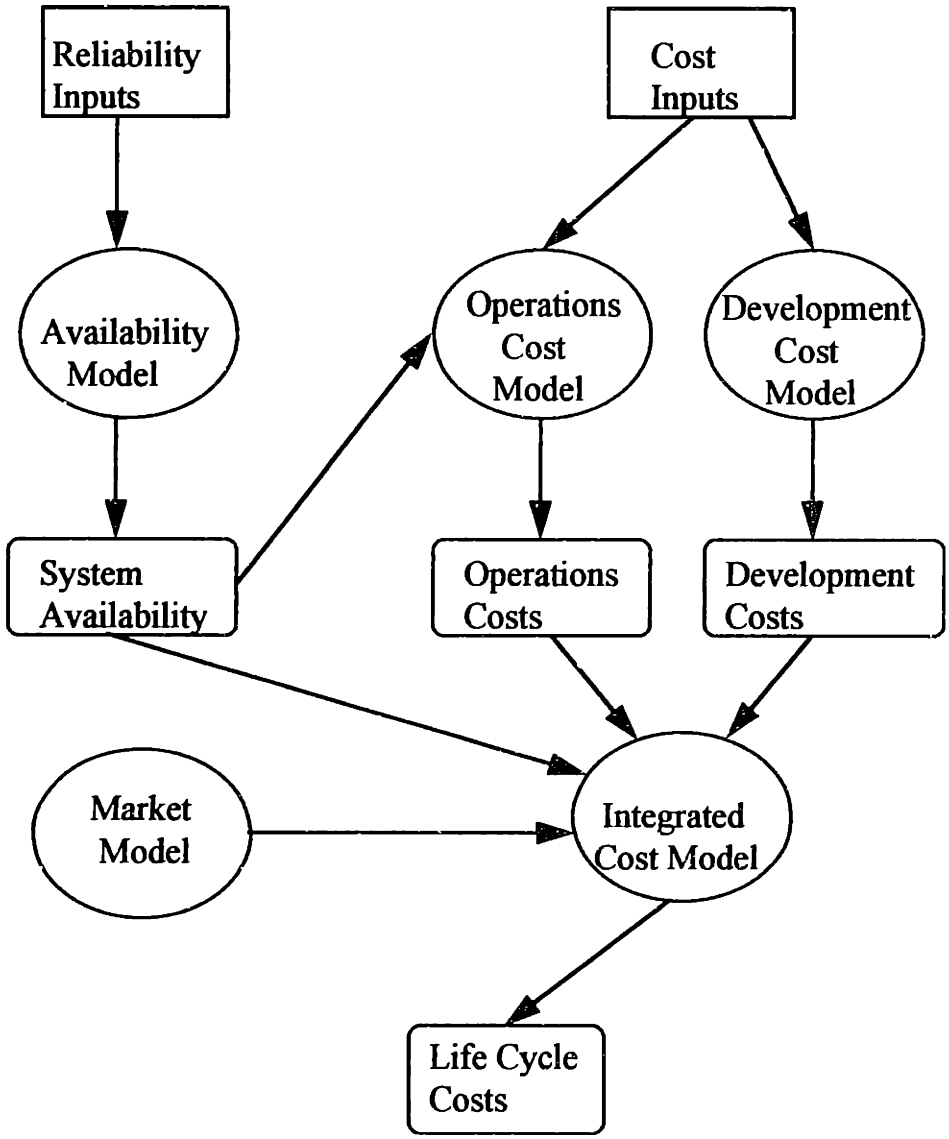


*Figure 2.3: Sub-Model Interactions*

## 2.4 Availability Modeling

The foundation for the calculation of the mission objective availabilities rests in the calculation of each function's availability. Figure 2.4 shows a basic state representation for a function. The function is in the *nominal state* when it is fully operational and there is

no immediate need for action by an operator or computer. The function transitions to the *action required state* when action is required by an operator. The transition back to the *nominal state* represents such a recovery action. In addition, the function may permanently fail with no ability to recover, which may result from a hardware failure or the total consumption of a non-replenishable resource such as propellant.

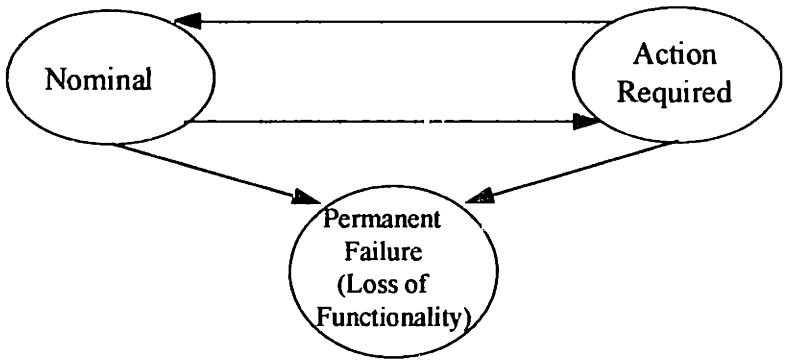


*Figure 2.4: Functional Availability Model*

The transitions to the *permanent failure state* are based upon hardware reliabilities. The transition to the *action required state* is based upon repairable failure rates, and scheduled operations frequencies. The transition back to the *nominal state* is determined by the human and/or computer repair rates and reliabilities.

## 2.5 Cost Modeling

As shown in Figure 2.3, the life cycle costs attributed to automation are separated into two main branches: development and operating costs; and opportunity costs which depend on the system availability. As automation is introduced, there is in general a need for additional software and possibly additional or different hardware. This will generally increase the development costs. However, as system functions are automated, the need for human interaction may decrease. This in turn would reduce the operations costs. Not only is the number of operators reduced, but there is also a reduction in support staff and overhead associated with these operators. However, if the software that has been implemented is unreliable, humans will be needed to resolve processor deficiencies.

## 2.6 Levels of Automation

A great deal of attention has been paid in recent years concerning the interfaces between machines and their human operators.[6] Of specific interest in the space community is the problem of remote operation. Despite the popularity of the topic, however, most of the efforts have been involved in isolating failure modes of the combined system and providing insight into control issues. The idea that automation can be introduced gradually is recognized, but definitions of discrete levels tend to be fuzzy.[6] The following section describes a more rigorous definition for discrete levels of automation, and what they mean in terms of task sharing between any two processors. These processors may be a human and a computer, or two computers.

Figure 2.5 illustrates the interaction among processors to perform a task or function. In general, each of three processors (human, ground computer, or OBP) may be involved in the completion of the task. The task or function to be performed is recognized by the space processor through sensors located throughout the spacecraft. The OBP may act upon this data, or may relay it to the ground computer. This computer may take action, or inform the human operator of the spacecraft status. If action is to be taken, the decision maker (human, ground computer, or OBP, depending on the level of automation) sends commands back up the chain to the OBP which then routes the command to the appropriate actuators.

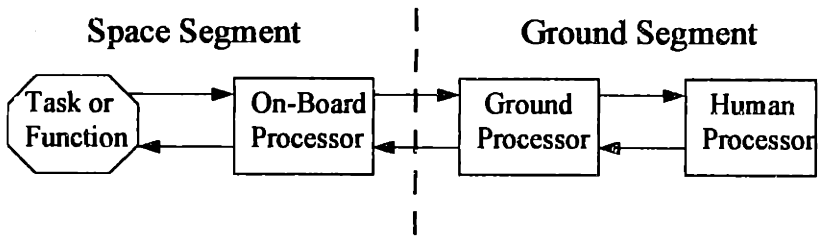


*Figure 2.5: Processor Interactions to Perform a Task*

Increasing the level of automation shifts responsibility from the human operator to the ground processor or spacecraft processor. As can be seen in Figure 2.5, there are two

processor interfaces to which automation may be applied. One such interface exists between the human and the ground computer processor. Automation may also be applied to the interface between the control center and the spacecraft processor. Therefore, the overall level of automation for a given task or function must be defined by each of the levels of automation at the two interfaces.

Regardless of the level of automation, one of the processors is generally responsible for performing the task, and is termed the *primary processor*. There may, in general, be cases during which the primary processor is unable to complete the task. This may be due to a hardware or software failure, human error, or to a state of the system for which the software was not designed. There also may be cases where the software has been designed to not attempt to perform the function under certain conditions. The inability of the primary processor to perform the task is termed a processor deficiency. A *secondary processor* may be defined to take over the function once a processor deficiency has occurred. A *tertiary processor* may also be defined to recover from a secondary processor deficiency.

One possible relationship among the primary, secondary and tertiary processors may be made more clear through an example. Suppose that a low Earth orbiting (LEO) satellite uses an Earth-Moon-Sun triangularization to determine its orbital parameters. Software has been written to do this on board the spacecraft, and thus the OBP is the primary processor for this function. Certain positions of the Sun, Moon, and Earth with respect to the satellite produce ambiguous results. In this case, a warning message is sent to the control center. The ground computer usually can resolve the ambiguity using recent telemetry and a model of the spacecraft dynamics. This software may be too complex and large to be run in space, and so it is kept on the ground. The ground computer attempts to resolve any situations which the space processor cannot. Thus, the ground computer is the secondary processor. Suppose that there are some situations for which the ambiguity cannot be resolved by the ground processor. In these cases, a warning light on the operator's display may be illuminated, and the human operator is then responsible for

resolving the problem. The human is then the tertiary processor. Only after both the space computer and ground computer are unable to determine the orbital parameters is the human operator brought into the loop to resolve the problem.

Automation of satellite systems may be accomplished by placing the responsibility for tasks on space or ground based computers rather than on a human operator. However, there are many cases where even in an "automated" system, the human is not completely out of the loop, but may monitor the computer processor. In order to preserve generality in the model, it is important to capture this subtlety of adding automation. To this end, discrete levels of automation have been defined which can describe the application of automation in an incremental manner. This allows a gradual transition from fully un-automated (human control) to fully automated (no human intervention). Each level of automation defines the degree to which each processor (human or computer) is responsible for the completion of a task.

The following sections describe the levels of automation associated with the satellite - control center interface (remote interface) as shown in Figure 2.6. The definitions for the levels corresponding to the control center - human interface are analogous to those for the remote interface.

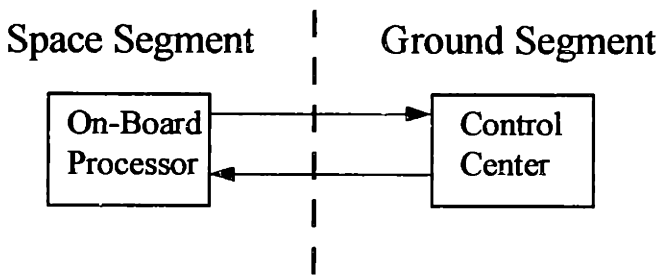


*Figure 2.6: Remote Link*

There are six levels of automation defined, ranging from fully automated to no automation. Each level represents a variation in the responsibilities and information passed between the space processor and control center. Thus, the assignment of primary

and secondary processors varies with the level of automation. Figure 2.8 shows the information flow and the processor assignments for the six levels, which are discussed in more detail below. A grayed-out processor indicates that for that level of automation, the processor is not used.

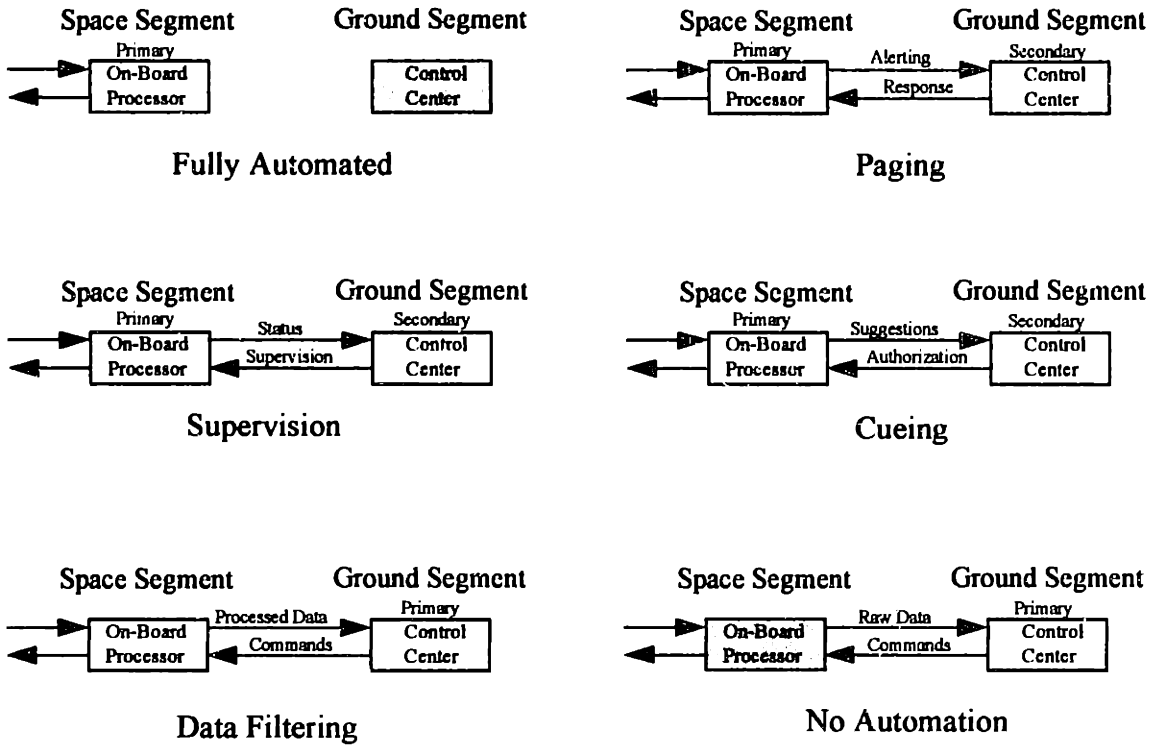


*Figure 2.8: Levels of Automation*

### 2.6.1 Fully Automated

Full automation implies that the space processor is the primary processor, and performs the function with no intervention from the ground control center. Since information regarding the task is not shared with the ground, if the space processor fails to correctly perform the task, the ground would not be notified. An example of a fully automated system is a "fire and forget" air to air missile. Once fired, not only is the missile fully responsible for finding and destroying the target, but the pilot is unable to enter the loop to ensure that the mission is accomplished.

### 2.6.2 Paging

With paging, the task is performed by the space processor, but the control center is notified at the occurrence of any failure or processor deficiency. Thus, the OBP is the primary processor, and the control center is the secondary processor. The major advantage of paging is that the control center is informed of problems as they occur, but dedicated ground computers or personnel do not continuously monitor the satellite. An example of a system with paging would be a modern automobile's computer diagnostics system. The computer checks and adjusts variables such as fuel-air mixture to compensate for poor performance, as indicated through the car's oxygen sensor. If the on-board system is unable to bring the oxygen reading within desired bounds by itself, the "service engine soon" light on the dashboard will light up. The mechanic must be paged, or, in this case, the car itself is "paged" and brought into the shop. Notice that until the car had trouble, the mechanic in the shop was fully unaware of the car's status.

### 2.6.3 Supervision

As with the levels of automation previously discussed, with supervision the task is nominally executed by the space processor (primary processor). However, the space processor's actions are monitored continuously by the control center (secondary processor). The ground may also be informed through an alarm if a primary processor deficiency should occur. The ground station may at any time assume responsibility for the task. When the human is the secondary processor, supervision effectively increases the probability of a timely response by the ground processor over the paging level of automation since the human would already be monitoring the control center. Also, since the ground is continuously monitoring the task, humans would be able to watch the evolution of events unfold, and therefore may be more likely to correctly interpret the reason the primary processor could not perform the task. Examples of a system operating under supervision might be an aircraft flight management system, or a nuclear power plant. Although a computer is responsible for maintaining the safe operation of the system, human operators are required to closely watch for deviations. This analogy also reinforces the idea that supervision may reduce the response time to a takeover once the

primary processor is unable to perform the task. Safety in a nuclear environment demands this response time to be small, and therefore supervision is used rather than a level of automation such as paging. With supervision, the secondary processor implicitly agrees with each of the primary processor's actions.

### 2.6.4 Cueing

At the cueing level, the system may require ground intervention at various points during the task's execution. The satellite would still attempt the task by itself, and therefore remains the primary processor, but must first obtain authorization from the ground. This is analogous to a PC prompting the user to verify any delete actions. Although this decreases the probability of incorrectly performing the function, it may also increase the time required to perform the task relative to the time which would be required if the task was more automated. Thus, increased availability may come at the cost of a more time consuming process. With cueing, the secondary processor must explicitly agree with the primary processor's actions.

### 2.6.5 Data Filtering

At this level of automation, the remote system is not responsible for performing the task, but aids the ground by filtering the downlinked information. Therefore, the control center is the primary processor. During a failure of the function (e.g. flight hardware failure), the satellite could send a more detailed set of data to the ground. This could tend to lessen the data load the ground must deal with, and hopefully improve the ease in which failures are identified and analyzed. The actual implementation of the function may still be done on the satellite because the function may require reconfiguration of hardware or software mounted on the satellite. A good example of data filtering can be found in many modern satellite operations systems (for example Integral System's Epoch 2000, or FIXIT [7]). Rather than scrolling raw telemetry to the screen, as was done in the past, some newer systems present the data in a graphical format such as system block diagrams which allow the data to be interpreted much more easily and accurately by a human operator.

### 2.6.6 No Automation

For this level, there is no pre-sorting of data before transmission to the ground. The data must be interpreted by the ground in raw form, and the task is fully performed by that processor. This may result in some development cost savings relative to a more automated system due to less software, but may increase operations costs due to the required operator workloads which may result from sorting the data on the ground.

## 2.7 Qualitative Effects of Automation

Ultimately, automation can impact the system through either its cost or its availability. Automation affects availability through the processor (software or human) reliabilities. The processor's reliability in performing a task is also affected by the characteristics of the processor and task complexity. Humans have different strengths than software in solving problems. While humans are generally more flexible, they may be less consistent. The reliability of a processor (human or software) is also dependent to some extent on the cost expended during its development. The following paragraphs present a discussion of such issues in a qualitative sense.

Figure 2.9 shows the qualitative relationship among software reliability, task complexity, and software development cost. Given a software provider and production methodology, the software's reliability can be increased mainly through increased testing which results in an increase in the development costs (as well as an increase in the delivery time). The reliability is also affected by the task complexity. For a given software cost, the software reliability will decrease as the task complexity increases.

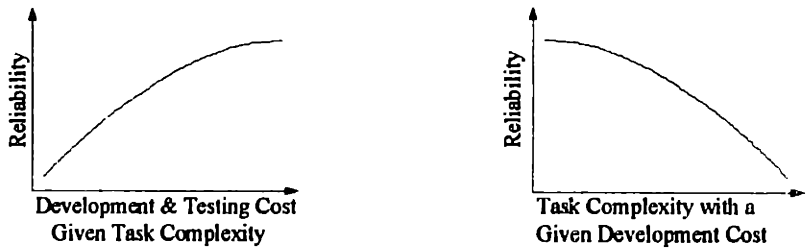


*Figure 2.9: Qualitative Relationships Among Software Reliability, Task Complexity, and Software Cost*

In general, the evaluation of software reliability will be a function of its development cost, task complexity, and level of automation as shown in Figure 2.10. Qualitatively, as the level of automation varies, the actual goals of the software may change, therefore impacting its cost and reliability. However, this variation is dependent on the overall operational strategy. For example, with Cueing, the flight code may be expected to obtain a suggested solution for every possible failure mode. Another approach might be to design the flight code to provide a suggestion only for certain failure modes, and to pass responsibility to a human for more complex or critical modes. Each approach will result in different cost and reliability characteristics because the "task" is actually changing slightly. The software model of Figure 2.10 may vary from one software provider to another, and may change as a function of the process used to develop the software.

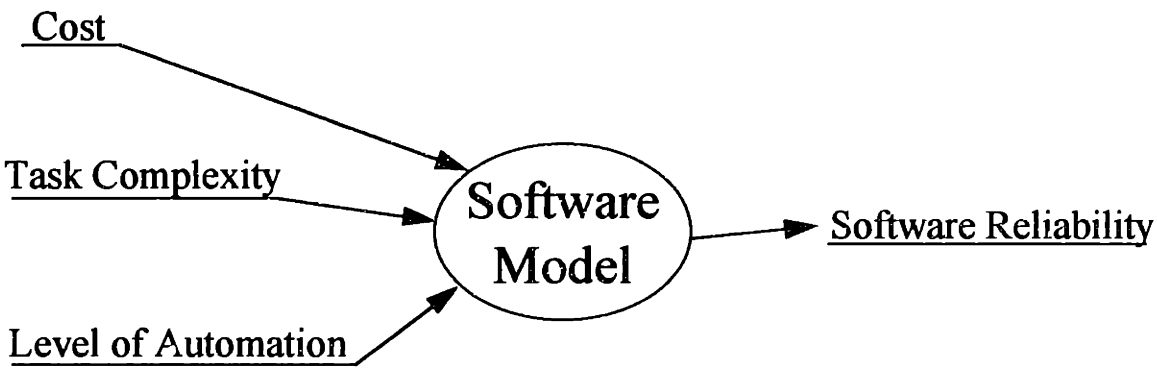


*Figure 2.10: Software Reliability Drivers*

The cost-reliability relationship for humans, shown in Figure 2.11, is similar to that of software. The human's training cost is analogous to the development cost associated with testing in software. This may be an increased salary required to hire more qualified personnel, or may represent an increase in training costs or improved facilities to enhance the performance of existing personnel. Human reliability as a function of the task complexity is also schematically shown in Figure 2.11. Unlike computers, human performance may decrease for very simple tasks.[8] Humans tend to become bored, inattentive, or careless for simple tasks. As the complexity increases, the human's attention sharpens, and fewer errors are made. At some level, however, the problem becomes complex enough that human reliability begins to decrease. The variation of human reliability with the level of automation is highly dependent on how the automation interfaces with the human. The display can affect the human's performance. In addition, environmental factors such as shift duration and lighting can alter the human's performance. An estimate of human reliability may require an experimental study of the proposed interfaces and may be case-specific.

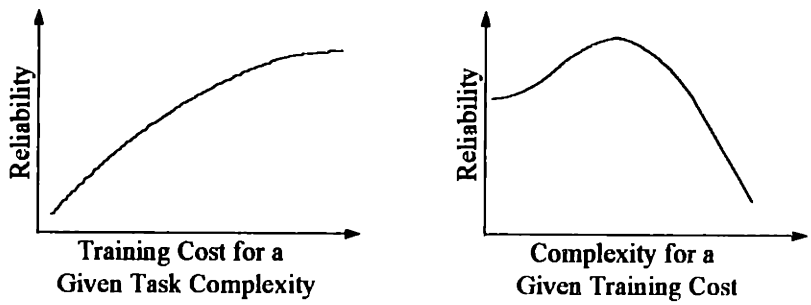


*Figure 2.11: Qualitative Relationship Among Human Reliability, Training Costs, and Task Complexity*

Note that the curves in Figure 2.9 and Figure 2.11 represent general trends. More formal models are needed to accurately estimate human and software performance and it is likely that such models will be highly dependent on the specific implementation.

This section illustrates the main drivers for human and software reliability. Thus, for a given task and level of automation, the cost and reliability are dependent primarily on each other. The costs and reliabilities for software and humans are then used as inputs to the methodology which determines the impact on the overall system through the mission objective availabilities and life cycle cost.

# 3. Availability Models

## 3.1 Events, False Events, Failures and Permanent Failures

Satellite operations are focused mainly upon two types of occurrences: events and failures as shown in Figure 3.1.

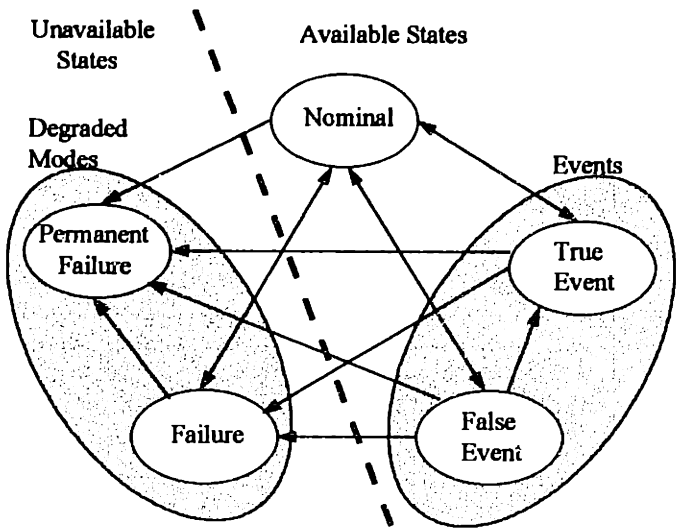


*Figure 3.1: Events, False Events, Failures and Permanent Failures*

The system transitions from a nominal state into the event state whenever action is needed (or perceived to be needed) to be taken in order to maintain the system's availability status. The corrective action, if successful, results in the system transitioning back into the nominal state. Some events result from planned actions, such as stationkeeping, or battery charging. These occurrences, termed *true events*, require some type of system action, but

do not result in the degraded performance of the system, and therefore do not directly impact availability. *False events* occur whenever sensor elements wrongly indicate an event has occurred and action must be taken, or when a processor performs an action when it is not needed. Presumably, if the false event is not recognized as a false event, some action will be taken. This action may be inconsequential, resulting in a transition back into the nominal state, or may result in a true event, or a failure, which is discussed below. Although the false event may not directly result in system unavailability, it may increase the operator workload.

*Failures* and *permanent failures* represent the second type of occurrence. A failure is an occurrence which degrades the performance of the system, but is repairable, and therefore requires some system action. The degraded performance means that the function is unavailable. Failures include anomalies, planned down-time of the system, or failures of primary units for which a backup exists. A separate failure mode may be defined to model each specific failure since different failure modes may be more severe than others. A permanent failure is one which results in the permanent loss of the function. This would include losses due to failure of both backup and primary units, and typically can be defined by the hardware reliability (including any redundancies). Since permanent failures occur due to natural hardware failures, transition into this state is possible from every other state. By definition, a permanent failure cannot be repaired, and the function will remain in this state forever.

## 3.2 Markov Modeling

The state representation of each function as shown in Figure 3.1, with probabilistic transitions among them, provides a basis for stochastic models. Since satellites are typically designed to be very reliable and contain redundant systems, it is desirable to choose a model that captures the characteristics of all probable failure modes without requiring extensive computation. Markov models not only capture the behavior of highly reliable systems, but can do so with significant savings in computational time compared to other modeling techniques such as Monte Carlo Simulations.[9] Run times can be large,

however, if the system model requires many states, although some techniques are available to reduce the number of states needed.[9,10] Additionally, Markov models are well suited to problems in which events occur sequentially (e.g., a failure event, followed by recovery steps to return to normal operation). Further discussion of Markov modeling can be found in Appendix A.

## 3.3 Functional Availability Model

The end goal of the functional availability model is to provide an estimate of each function's availability over time. The calculation accounts for planned and unplanned operations (event and failure recoveries) and includes failures and events induced by operations on other functions (functional interdependencies). To do this, a Markov model such as that of Figure 3.1 is defined for each function.

In general, each state transition in the model consists of two parameters: the success probability and the mean time to transition. Two parameters are used because there is some probability that the transition will be successful (success probability), and whether the transition occurs or not, there is some mean time for the transition to take place. The existence of the success probability in the recovery transition results from two possible outcomes of the recovery attempt. If the recovery is successful, the function transitions to the nominal state. If not, it remains in the failure or event state. For failures, there is no such option. Only a mean time to failure (failures are inevitable) is required. The success probability and transition time are combined to form the transition probability, P(tx), as shown in Eq. 3.1

$$P(tx) = P(\text{success}) \cdot \left[ 1 - e^{-\left(\frac{dt}{mttx}\right)} \right] \tag{3.1}$$

where P(success) is the success probability, dt is the time step, and mttx is the mean time to transition. For failures and events, P(success) is assumed to be 1, and the transition

probability is then a function only of the time step and the mean time to failure or mean time to event.

The transition to permanent failure is assumed to be caused by hardware failures (age), and therefore is assumed to be independent of the current state. Since the permanent failure state can be entered, but not exited, it acts as a sink in the model. In the absence of this state, the model, after a transitory stage, would stabilize upon a steady state probability vector where the net flow into each state is equal to the net flow out. Therefore, the availability model can be solved first for the steady state value, which determines the relative state probabilities. The probability of permanently failing is modeled as an exponential distribution:

$$P(\text{perm. fail.}) = 1 - e^{(-t/mtpf)} \tag{3.2}$$

The remaining state probabilities are then determined by "diluting" the steady state values by the probability that a permanent failure has occurred. This method greatly simplifies the calculations since the steady state value can be determined shortly after the system beginning of life, and the probability of experiencing a permanent failure up to time t is given by Eq. 3.2.

Figure 3.2 shows the functional Markov model without the permanently failed state, and with multiple failure modes.

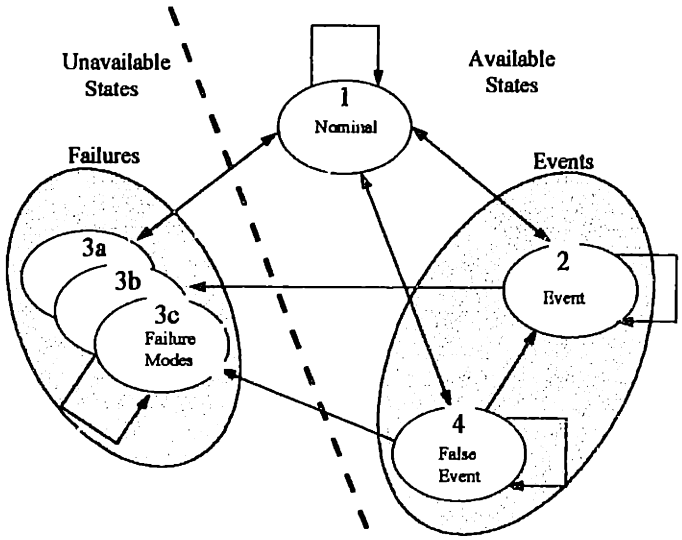


*Figure 3.2: 4-State Functional Availability Markov Model*

### 3.3.1 Overview

The model in Figure 3.2 uses the function's mean time to event, mean time to false event, mean time to failure for each failure mode, and mean time to permanent failure to calculate transitions from the nominal state. The level of automation, along with human and computer processor reliabilities and execution times are used to estimate mean times to repair for events and failures.

### 3.3.2 Recovery Process

The transitions to the nominal state in Figure 3.2 are called recovery transitions, and represent a recovery from an event or failure. These transitions may be further broken down into a string of states or recovery processes shown in Figure 3.3. Once the event has been detected, it must be communicated to a decision maker. This decision maker must determine the actions that must be taken to bring the system back into its performance envelope and then implement this action.

The probabilities and times to transition from one state to another in Figure 3.3 may differ for each type of recovery (failure, event, or false event). The overall recovery transition may be derived from the individual recovery process transitions as shown in Appendix B.

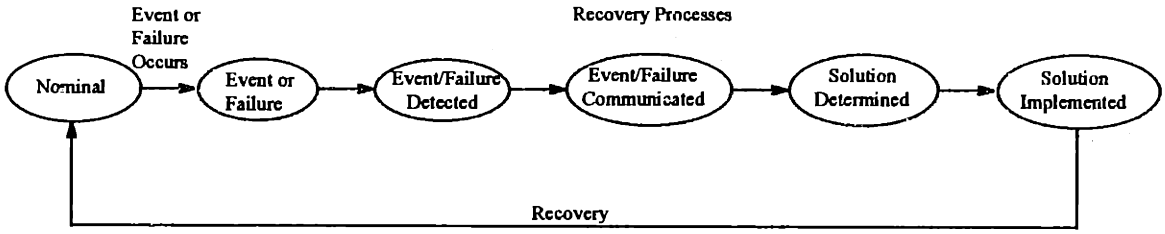


*Figure 3.3: Event/Failure Recovery Process*

The probability that an event occurs, as well as the transition probabilities and transition times from one state to the next in the Recovery Process, are dependent on the level of automation. For example, whether or not a human is in the loop will affect the probability that a solution is determined. In general, each transition in the process requires a processor element and, if automated, software or hardware. Depending on the level of automation, the transition probabilities can be determined using statistical data on human reliability and fault trees to determine the probability that all the required components are functioning.

### 3.3.3 Analysis Neglecting Dependencies

The Markov model transition probabilities are calculated based upon the level of automation as described in Appendix C. Once the transition probabilities have been determined, a reduced Markov model is generated by "collapsing" the larger models which may contain multiple failure modes. This information is then used to determine the function's independent steady state probability vector as well as the overall mean times to recovery from failures or events.

Each transition is calculated from knowledge of the performance of the processors involved (operators and computers) as well as the level of automation governing the recovery process. The equations contained in Appendix C are used to calculate the overall transition success probabilities as well as the transition times. The logic underlying

the transition equations is discussed in the following paragraphs. Although the explanations refer to the space processor and ground processor for each level of automation, an analogous relationship exists for the ground levels of automation.

### *3.3.3.1 Recovery Process Transitions*

Table 3.1 provides the general form of the success probability and transition time equations for the various levels of automation. The equations of Table 3.1 assume that the software performing the interface tasks (e.g. paging, data filtering, etc.) are perfectly reliable and instantaneous. Also, they refer to only the remote level of automation. Appendix C contains the detailed equations including interface task performance. In the equations of Table 3.1, P represents a success probability, mtt represents a transition time, and the subscripts s and g represent the space processor and ground processor respectively.

*Table 3.1: Basic Form of Transition Equations*

| Level of Automation | Success Probability | Transition Time |
|---|---|---|
| Fully Automated | $P_s$ | $mtt_s$ |
| Paging | $P_s + (1\text{-}P_s)P_g$ | $\dfrac{1}{\left[\dfrac{P_s}{mtt_s} + \dfrac{(1-P_s)}{(mtt_s + mtt_g)}\right]}$ |
| Supervision | $P_s + (1\text{-}P_s)P_g$ | $\dfrac{1}{\left[\dfrac{P_s}{mtt_s} + \dfrac{(1-P_s)}{(mtt_s + mtt_g)}\right]}$ |
| Cueing | $P_s + (1\text{-}P_s)P_g$ | $mtt_s + mtt_g$ |
| Data Filtering | $P_g$ | $mtt_g$ |
| No Automation | $P_g$ | $mtt_g$ |

The following paragraphs describe the logic behind the equations of Table 3.1 for each level of automation.

The fully automated system relies solely on the space processor to perform the recovery tasks. Therefore, the success probability for the transition is the success probability for

the space processor, and the mean time to transition is the mean time to completion for the space processor.

Paging allows the ground processor to be informed of the progress of the space processor. Therefore, the system will correctly perform the task if either the space processor is successful, or, in the case of a space processor failure, if the ground processor is successful. Success is then defined by either the space processor alone or both the remote interface software and ground processor succeeding. The transition time will be a weighted combination of the time required if the space processor is successful and the time required if the space processor is not successful (fails). The details of such a weighted combination of transition times is included in the Appendix C.

The supervision level of automation follows the same general algorithm as paging, but the ground processor success probability and transition time may not be the same as with paging.

Cueing requires the space processor to complete the task, and the ground processor (human or computer) to authorize the action. This assumes that the ground actually checks the results of the processing, which may take some time, but this may not be as much as if the ground processor were to complete the task alone. Nevertheless, only one processor need be correct.

For Data Filtering, the space processor is no longer responsible for the determination of the solution for the recovery (although it may be required for sensor detection, communication of the failure or event, or implementation of the solution as well as the automation tasks (in this case data filtering). The ground processor performs the function, but is provided pertinent information by the space processor.

With No Automation, only the ground processor is required to perform the task. Since there is no automation aiding in data reduction, the mean time to completion for the task

may be greater if the human is performing the recovery. However, the absence of automation also removes a potential source of failure during the recovery process, and therefore may increase success reliability. The converse may also be true, however, if the data are so complex or ambiguous that humans often misinterpret them.

### *3.3.3.2 Accounting for Multiple Failure Modes*

One or more failure modes may be defined for each function. However, by the definition of a failure in section 3.1, all failure modes are assumed to result in the unavailability of the function. Therefore, for the purpose of calculating the function's availability, they may be combined together to a generalized failure state as shown in Figure 3.4. Once the failure modes have been combined, all functions can be represented by the same 4 state Markov model.

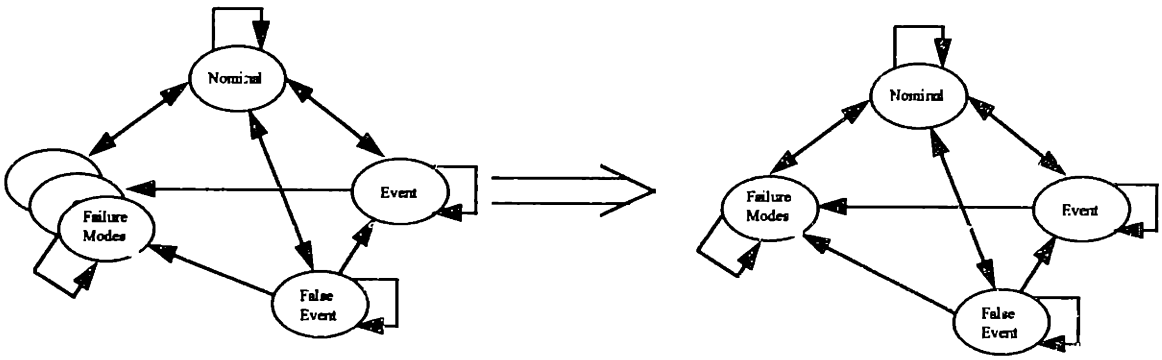


*Figure 3.4: Collapse of Multiple Failure Modes*

The total mean time to failure is determined by the aggregation of transitioning into each failure mode, and is given by Eq. 3.3,

$$\frac{1}{mtf} = \sum_i \frac{1}{mmtf_i} \tag{3.3}$$

where mtf is the overall mean time to failure and $mmtf_i$ is the mean time to failure of the $i^{th}$ failure mode. The derivation of Eq. 3.3 follows from the same logic of the derivation of weighted transition times as given in Appendix D.

Eq. 3.3 is used to calculate the an overall mean time to failure to *any* failure mode. Therefore, it is desirable to calculate an overall mean time to recovery from any mode. This will allow the states representing the separate failure modes to be collapsed into a single state in the Markov model. However, an expression such as Eq. 3.3 cannot be used to combine recovery rates. This is because the overall recovery time is dependent on how often each mode must be recovered from. The overall mean time to recovery must then be derived from the steady state probability vector and the overall mean time to failure.

The Markov analysis which includes all failure modes results in a steady state probability vector with elements representing the probability of being in each failure mode as well as the probabilities of being in the event, false event, and nominal states. The overall failure state probability can then be calculated by the sum of the state probabilities over all failure modes. The state probabilities for the nominal, event, and false event states remain unchanged between the full and collapsed models. Since the mean time to failure is independent of the state, the three operational states (nominal, event, and false event) may be combined to form an available state as shown in Figure 3.5. As with the collapse of the failure mode states, the state probability for the available state can be determined by the sum of the state probabilities for the nominal, event, and false event states. The unavailable state is the collapsed failure state.

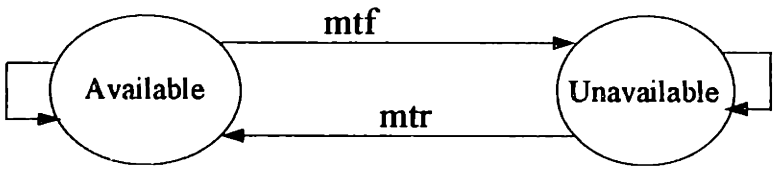


*Figure 3.5: 2 State Collapsed Functional Markov Model*

As pointed out in section 3.1, with the absence of permanent failures in the model, the probability state vector will reach a steady state value after a transitory period. The steady state probability vector is known from the Markov analysis of the un-collapsed model.

Also known is the mean time to failure from Eq. 3.3. Thus, the overall mean time to recovery from a failure can be calculated by Eq. 3.4,

$$\frac{mtr}{mtf} = \frac{P(\text{Unavailable})}{P(\text{Available})} \tag{3.4}$$

where mtr is the overall mean time to recovery, mtf is the overall mean time to failure, P(Unavailable) = 1 - P(Available) and is the steady state probability of the function being unavailable, and P(Available) is the steady state probability of the function being available. Further details regarding the derivation of Eq. 3.4 can be found in Appendix E.

### 3.3.4 Including Dependencies Among Functions

Dependencies among functions (functional interdependencies) arise whenever the operation of one function depends on the current state of another. Figure 3.6 shows a system with dependencies depicted as arrows from one function to another.

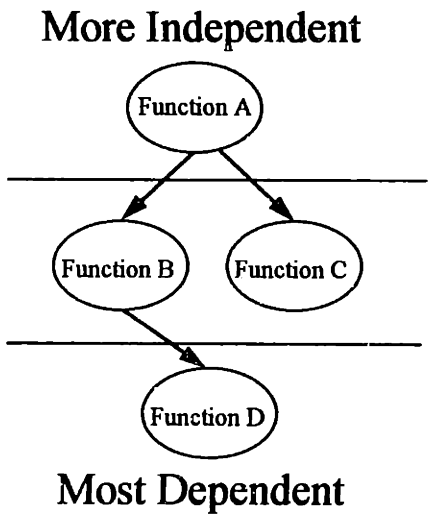


*Figure 3.6: Dependencies Among Functions*

As an example of dependency, the communications function of a satellite may transition to a failed state more quickly if the attitude control function is currently in a failed state than if it were in a nominal (available) state. The failure transition for the model in Figure 3.7

would use mtf1 if the attitude control function were in an available state, and mtf2 if it were in a failed state. Functional interdependencies of this type are usually represented in Markov models by creating a single model containing the states for each function as well as combinations (e.g., function A failed, and function C nominal) as shown in Figure 3.8. For systems with a large number of functions, this can lead to huge, complex Markov models which need to be truncated for practical computations.

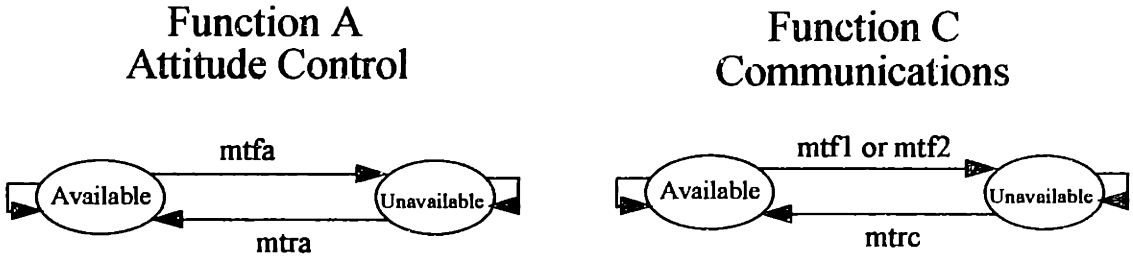


*Figure 3.7: Communications Function's Dependency on the Attitude Control Function*

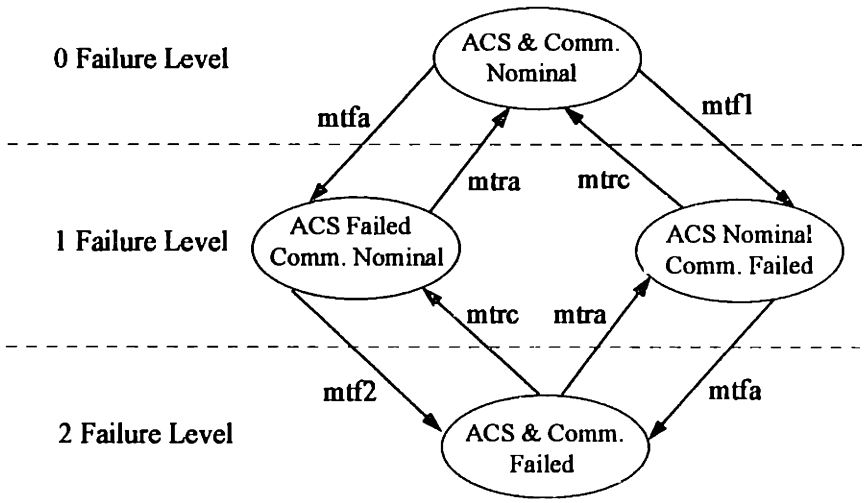


*Figure 3.8: Conventional Multi-Function Markov Model Showing Dependency of Communications Function on Attitude Control Function*

An alternative to this approach assumes that the mean time to failure is either the "nominal" mtf, or is set to zero. In such a case, as long as the attitude control function was in the available state, the communications function's mtf would be the nominal mtf (mtf1). As soon as the attitude control function transitioned to the failed state, however, the communications function would instantaneously transition into a failed state. Such a dependency implies that the communications function *requires* the attitude control function to be available in order to itself be available. If such requirements only exist in one direction (function A requires function B, but function B does not require function A), the dependency is *unidirectional*.

If all of the functional interdependencies in a system are unidirectional, each function may be modeled separately. This allows a hierarchy to be constructed with the functions at the top being independent of those below them. Thus, dependency flows up in the hierarchy, with each function only being dependent on functions which lie above them. This concept is explained further in this section.

Functional interdependencies are grouped into four types. Each type represents how the current state of an independent function can trigger a transition of the dependent function. Thus, an event or failure of the independent function (function A) always results in an event or failure of the dependent function (function B). This is shown in Figure 3.9.

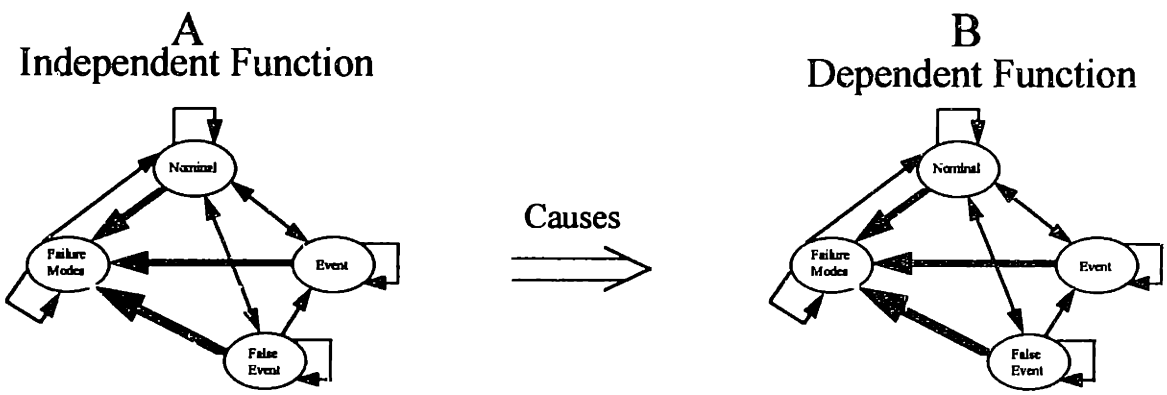


*Figure 3.9: Example Functional Dependency*

Figure 3.9 illustrates a functional dependency. In this case, the failure of the independent function results in the failure of the dependent function. This is a "type 4" dependency as defined in Table 3.2. The following paragraphs describe each type of dependency in more detail in terms of some occurrence of the independent function causing an occurrence of the dependent function.

Table 3.2 summarizes the dependency types along with their definitions. The implication of each dependency type is discussed in the following paragraphs. Functions with very weak dependencies (the transition probabilities do not change much with the independent function's state) may be assumed to be independent. If there is sufficient doubt of the degree of dependency, the system may be modeled with no dependency and with one of the dependency types and the results compared. This may give some indication to the implications of the assumption of no dependency.

*Table 3.2: Functional Dependency Types*

| Type | Occurrence in Independent Function | Result in Dependent Function |
|---|---|---|
| 1 | Failure | Unavailability |
| 2 | Event | Event |
| 3 | Failure | Event |
| 4 | Failure | Failure |

A type 1 dependency is a "soft" dependency in that as long as the independent function is in a failed state, the dependent function will not be available to perform an objective. However, when the failure of the independent function is resolved, the dependent function immediately becomes available with no operator or processor intervention. The relationship between the lights and power supply in a building is an example of a type 1 dependency. Without power to the building, the lights will not go on. However, as soon as power is reinstated, the lights which were on before the failure will be back on.

Type 2 dependencies are "hard" dependencies. Each occurrence of the event of the independent function causes an event in the dependent function. This type of dependency may exist for the stationkeeping and ephemeris determination functions of a satellite. Each time a stationkeeping maneuver is executed, the ephemeris buffer of the spacecraft must be updated. Both functions are available since the satellite is still within its orbital slot, and the ephemeris buffer always contains data.

In a Type 3 dependency, an event in the dependent function is triggered by a failure in the independent function. This type of dependency is common in failures which require reconfiguration of the system. The failure of a primary unit would cause the secondary unit to be switched on (failure of the independent function), but a ground-maintained database may need to be updated to reflect the changes (event of the dependent function). Here the independent function is defined by the operation of the primary unit while the dependent function is the database maintenance function. Note that until the redundant unit is switched on, the function it is responsible for is unavailable, while the database is available throughout the process.

In a type 4 dependency, a failure of the independent function causes a failure in the dependent function. This may result from a power regulation failure (failure of the independent function) blowing an amplifier which has a backup (dependent function). Until the power regulation function is fixed, the backup will not be available.

The functional interdependencies impact the overall model in two ways. The interdependencies which cause unavailabilities either by soft dependencies or by causing failures impact the availability of each function by adding additional failure modes to the dependent function. All hard dependencies affect the operations of the system by increasing operator workloads due to additional events and failures which only occur as a result of the system interdependencies. This section will only deal with the availability effects. Operations effects of dependencies are discussed in section 3.5.

All dependencies are assumed to be unidirectional. A bi-directional absolute dependency would cause both functions to become permanently unavailable upon the failure of either. Function A's failure would cause function B to fail as well, but as soon as function A was repaired, it would fail since function B was still failed. Bi-directional dependencies may result from trying to model two or more highly coupled functions separately. If their failures are all dependent on each other, and their recoveries must take place simultaneously, they probably can be modeled as a single function. An example may be the coupling among attitude control axes. If each axis affects the others, it would be more appropriate to model them with a single function rather than one for each axis of control.

The unidirectional assumption results in a hierarchy of dependence as shown in Figure 3.10. Note that Functions A, B, and C in the figure are independent of all other functions. Functions D - G are only dependent on Functions A, B, or C, and Functions H and I are Dependent on Functions A - G.

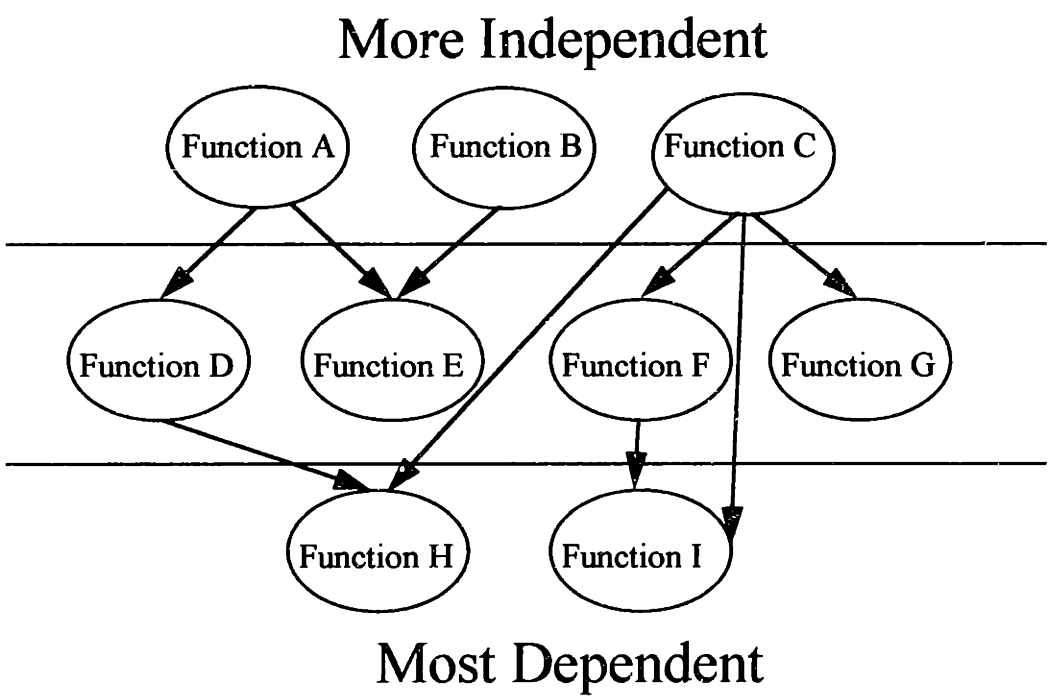


*Figure 3.10: Functional Interdependence Hierarchy*

Type 1 and type 4 dependencies result in the unavailability or failure of the dependent function. Therefore, the failure of the independent function is effectively an additional mode of failure for the dependent function. Therefore, the dependency can be incorporated into the dependent function's Markov model by adding a new failure mode. The mean time to failure for the new mode is equal to the overall mean time to failure of the independent function. With a type 1 dependency, the mean time to recovery for the new mode is equal to the mean time to recovery of the independent function alone since no failure of the dependent function has occurred (only its unavailability). For type 4 dependencies, the dependent function actually fails, and therefore the mean time to recovery for the new mode is the sum of recovery times for both functions.

The hierarchy of type 4 and type 1 dependencies is used with the independent analysis results to determine the actual functional availabilities accounting for the dependencies. For those functions on the top (independent) level of the hierarchy, the independent results still hold since these functions are not dependent on any others, and therefore have no additional failure modes due to dependencies. For those functions in the next level, additional failure modes are added to the dependent functions' Markov models, and the Markov analysis described in section 3.3.3 is redone. This results in a "corrected" overall mean time to failure and probability state vector. As the correction process moves down the hierarchy, the corrected results of higher (more independent) functions are used to add failure modes to those functions on the current level.

Note that the dependencies must be carefully defined. Refer to Figure 3.10. The calculation for Function I would begin with the addition of two new failure modes, one which results from a Function C failure, and another which results from a Function F failure. However, this is not correct. The failure rate of function F already accounts for Function C failures, so adding both failure modes would account for failures of function C twice. For such cases, only a single failure mode would be added to Function I's model, which accounts for failures of Function F. This can be made more clear by Figure 3.11 which shows a Venn diagram for the failure states of I, C, and F. I failures consist of I

alone (independent failures), and Failures caused by F (I and F). F failures consist of independent failures (which also cause I to fail) and those caused by C (I, F, and C). C failures only consist of independent failures (which also cause I and F to fail). Since the set of F failures includes all failures caused by C, the calculation of I failures does not need to explicitly account for the C failures.

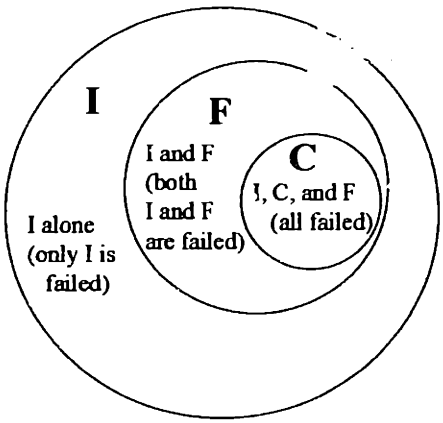


*Figure 3.11: Venn Diagram of the Failure States of Functions C, F, and I*

Several assumptions have been made in this section. One is that the bi-directional failures have been removed. This is most appropriately done during the functional decomposition before any analysis is completed. For example, if the roll angle control function and pitch angle control function's failures cause failures in the other, the two should be combined as a single function (roll/pitch control) with failure modes corresponding to roll failures and pitch failures. The second assumption is that the probability of two functions with an interdependency being independently failed is small. In other words, for every two functions with an interdependency, the product of their independent failure state probabilities is very small (negligible). This is essentially the same assumption which governs the definition of the multi-failure mode model in the first place. If the probabilities of being in each failure mode are small, the probability of being in both simultaneously is negligible, and the probability of being in any failure mode is essentially the sum of the probabilities of being in each mode. This assumption is essentially forced

since the probability of transitioning from one failure mode to another directly is assumed to be zero.

## 3.4 Objective Availability Models

The satellite / control center availability model uses the results of the independent functional availability analysis, the functional dependencies, and definitions of the satellite objectives and control center objectives. These lower level objectives combine to meet the mission objectives of the system. The satellite / control center objective level is a useful initial step in defining mission objectives which require several satellites.

### 3.4.1 Satellite and Control Center Objective Availability Model

The inputs to the model include the function independent availabilities, interdependencies, each satellite objective's functional requirements, and each control center objective's functional requirements. The model uses this information to calculate each satellite objective's availability and each control center objective's availability.

The model requires that the satellite and control center objective functional requirements are defined by a set of functions which are required to be available in order for the objective to be available. In general, a satellite or control center objective may be met through several sets of functions. This results from functional redundancy in the system, and is shown schematically in Figure 3.12. In this example, the objective can be met either by functions A, B, C, and D, or by functions E and F. These groupings of functions, characterized by the AND gate, are called redundancy strings. The OR gate means that any one string can provide the objective availability and represents some degree of functional redundancy in the system..

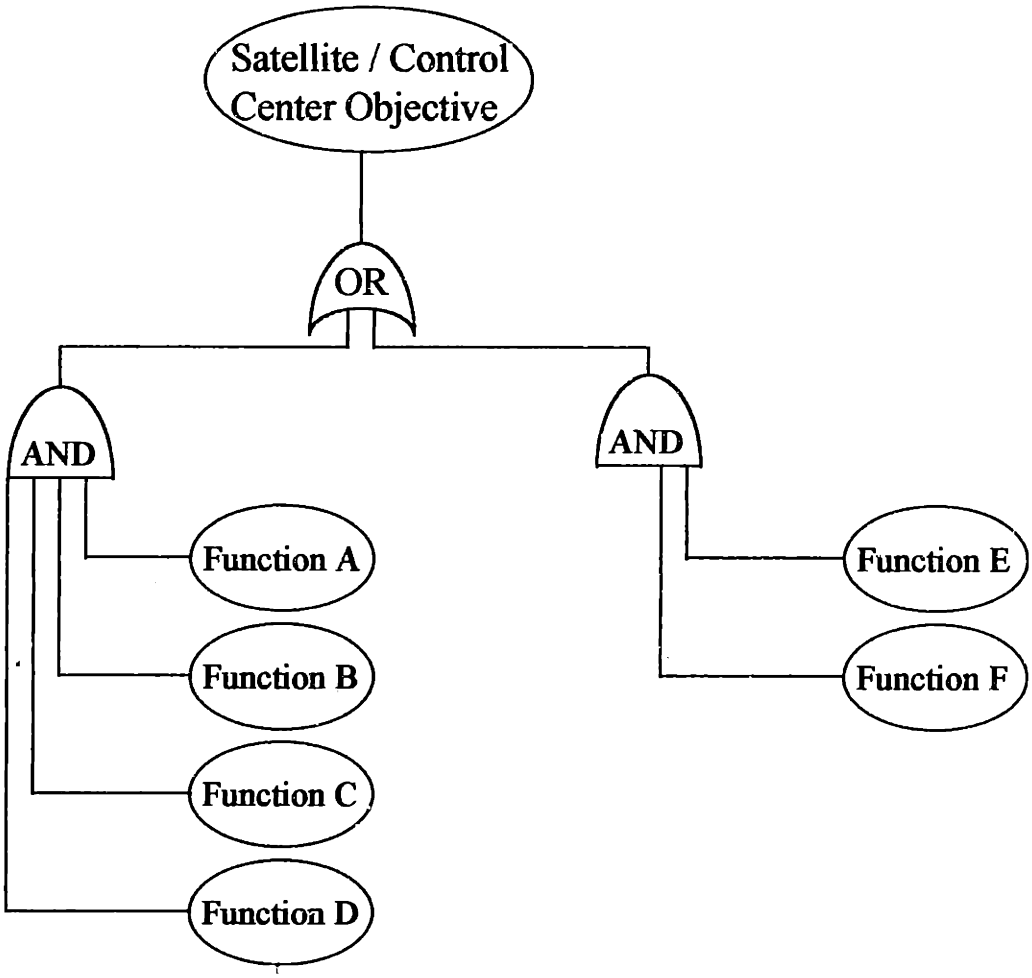


*Figure 3.12: Functional Redundancy in Satellite / Control Center Objectives*

The functions which make up each string is a set of "required" functions. When defining this set of required functions, functions which cause type 1 and 4 dependencies (which result in the dependent function's unavailability) of the required functions should also be included. For example, suppose that one string requires functions A and B. Also, functions A and B have a type 4 dependency on function C, and function C has a type 1 dependency on function D. The redundancy string should then include functions A, B, C, and D. The independent availability results are then used to calculate the overall availability of the redundancy string.

By retracing the dependencies of functions A and B to include all functions upon which they depend, a list of functions which can cause the unavailability of the objective is

created. This is similar to defining independent failure modes, but now for an objective rather than another function. The complete list of functions includes all functional failure modes which can result in the objective's unavailability.

Once the satellite and control center objective functional redundancy strings are expanded to include all functions upon which the original required functions were dependent, and the strings are checked for repeated functions, the calculation of the objective availability follows naturally. The AND operator means that all functions under it are required to be available. Thus, the availability of each functional redundancy string is determined by Eq. 3.5

$$A(RS_j) = \prod_i A(func_i) \tag{3.5}$$

where $A(RS_j)$ is the availability of the $j^{th}$ redundancy string, and $func_i$ is the $i^{th}$ required function of that redundancy string. Because dependencies are accounted for in the redundancy strings, A(func) is the function's *independent* availability including permanent failures. Up to this point, the analysis had neglected permanent failures. This has resulted in the calculation of independent and dependent steady state availabilities. These availabilities must be modified to include permanent failures, as shown in Eq. 3.6

$$A(func_i, t) = A_{ss}(func_i) \cdot e^{-\left(\frac{t}{mtpf}\right)} \tag{3.6}$$

where $A(func_i, t)$ is the true availability of function i at time t, $A_{ss}(func_i)$ is the steady state availability of function i, t is the time elapsed since the beginning of life, and mtpf is the mean time to permanent failure for function i. Eq. 3.6 can be used with either the independent or dependent steady state availability, yielding the corresponding actual availability.

An OR operator means that one or more of the redundancy strings needs be available to meet the objective's availability requirements. This is more easily stated mathematically as one minus the probability that all redundancy strings are unavailable, as shown in Eq. 3.7 where $A(O_k)$ is the availability of objective k, and $A(RS_j)$ is the $j^{th}$ redundancy string's availability.

$$A(O_k) = 1 - \left\{ \prod_j \left[ 1 - A(RS_j) \right] \right\} \tag{3.7}$$

### 3.4.2 Mission Objective Availability Model

The mission objective availability model uses the satellite objective, control center objective, and function availabilities, along with the satellite objective and control center objective availabilities to calculate each mission objective's availability.

Figure 3.13 shows an example system's mission, satellite, and control center objectives as well as the functions that support them.

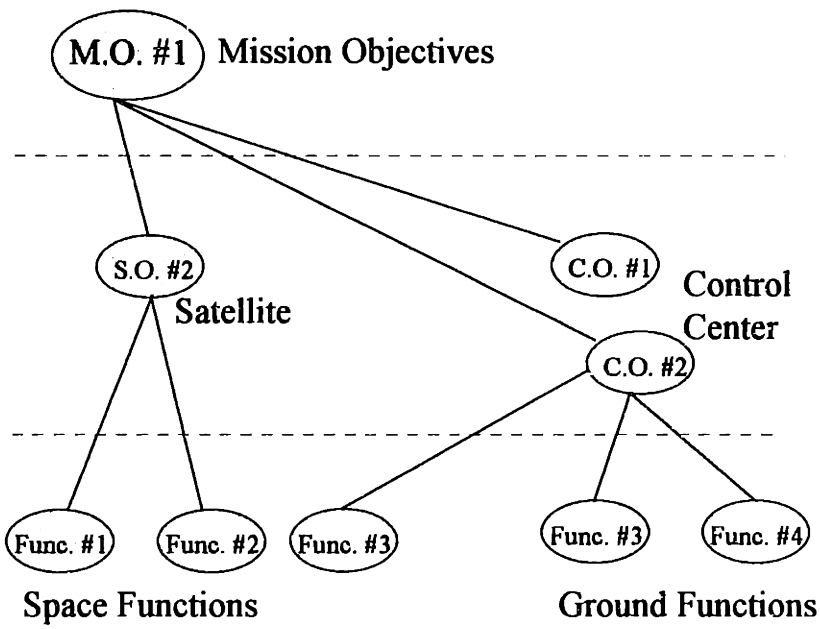


*Figure 3.13: Mission Objective Requirements*

The lower portion of the figure is similar to Figure 3.12. The satellite and control center objectives are met through some required functions. The mission objectives are then met through satellite and control center objectives. Thus, the mission objective availabilities are defined by the product of the availabilities of its required satellite and control center objectives. Thus, the mission objective availability, A(MO) is defined by Eq. 3.8. As with the definition of the satellite / control center objectives through redundancy strings, the satellite and control center objectives must be independent for the calculation of the mission objective to be correct.

$$A(MO) = \prod_{k} A(O_k) \tag{3.8}$$

## 3.5 Operations Staff Requirements

The functional availability model resulted in the calculation of each function's independent and dependent steady state probability vector, as well as the availabilities including permanent failures as was defined by Eq. 3.6. The calculation of objective availabilities used the true independent availabilities to avoid common failure modes being accounted for more than once. Operations requirements are calculated at the functional level, and therefore rely on the dependent availabilities. Since permanent failures reduce the probabilities of being in the other 4 states, the steady state availabilities are used to calculate the maximum operational requirements (once the function is permanently failed, events and recoverable failures will not occur, and therefore will require no operations). This model neglects operations due to work-arounds once a function has permanently failed.

Figure 3.14 shows the operations requirement inputs. The total operations requirements are due to event recovery (events and false events), supervisory requirements, and failure recovery. The dependent availability results already include failure-causing dependencies, and therefore can be used directly. However, event causing dependencies, (type 2 and type 3) were not included in the dependent analysis results since they do not affect the

function's availabilities and therefore the mission objective availabilities. The supervisory requirements account for operators performing monitoring roles required for lower levels of automation (below paging). During this time, the function is in a nominal state and does not require an operator to be actively doing anything, except monitoring the function's health to ensure all is well. This supervisory workload is independent of the function's failure and event rates and is provided as a model input.

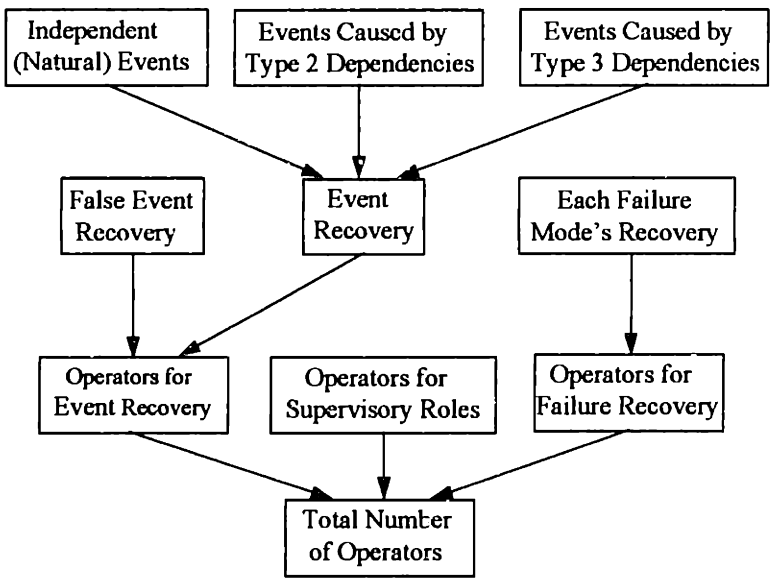


*Figure 3.14: Sources of Operations Requirements for Each Function*

Based upon the level of automation, the probability of a human operator being involved in each recovery (event, false event, or failure) at any given point in time can be calculated based upon the probability of the function being in the event recovery state (see Appendix F). In addition to availability data, the user must supply for each function's event recovery, false event recovery, and failure mode recoveries the overall time required for the human to complete the task, and the logged time to complete the task. The logged time is the actual time that the operator spends to complete the task. The overall time is the total elapsed time required for the human to complete the task and is equal to the sum of the logged time and any "dead" time during which the recovery is still ongoing, but the human is not actively involved. For practical purposes, logged time is that time during which the operator cannot perform any other duty except the recovery process.

### 3.5.1 Operations Requirements for Failure Recovery

Since functional interdependencies have already been incorporated into the model, the steady state probability of being in each failure mode is already known from the availability analysis. The number of operators required to support each failure mode's recovery operations, $N_{Fm}$, is then given by Eq. 3.9,

$$N_{F_m} = \left(\frac{LT}{OT}\right) \cdot H \tag{3.9}$$

where H is the probability that a human is performing a recovery process, and is calculated based upon the level of automation, the failure mode state probability, and the equations of Appendix F. LT represents the total logged time in man-hours, and OT represents the overall time in hours. The total number of operators for failure recovery of each function is then calculated by the sum over all failure modes as given by Eq. 3.10.

$$N_F = \sum_m N_{F_m} \tag{3.10}$$

### 3.5.2 Operations Requirements for Event Recovery

The probability of each function being in an event recovery or false event recovery state has also been calculated in the availability analysis. Therefore, the number of operators required to perform independently occurring events and false events (not including dependencies), $N_{EI}$, is determined by Eq. 3.11,

$$N_{EI} = \left(\frac{LT}{OT}\right) \cdot H \cdot \left(\frac{A_d}{A_i}\right) \tag{3.11}$$

which is similar to Eq. 3.9, but with a correction factor, where $A_d$ is the function's availability accounting for dependencies, and $A_i$ is the availability assuming functional independence. The calculation of H in Eq. 3.11 uses the event recovery level of automation and event recovery human operator performance. The ratio of the function's

dependent availability to the independent availability accounts for the fact that the probability of being in an event state had been calculated from the independent analysis, and up to now has not been adjusted for interdependencies, but dependency-induced failures decrease this probability in reality. The ratio assumes that the relative state probabilities for un-failed states (nominal, event, false event) should remain unchanged with an increase in the failure state probability. The ratio of dependent availability to independent availability reflects this by adjusting the probability of being in an event state (contained in the H variable) such that the relative probabilities of un-failed states remains constant, while the sum of state probabilities remains equal to unity. This is shown graphically in Figure 3.15. Notice that the relative magnitudes of the un-failed states remains the same, while their absolute magnitudes are adjusted for the increased failed state with dependencies accounted for.

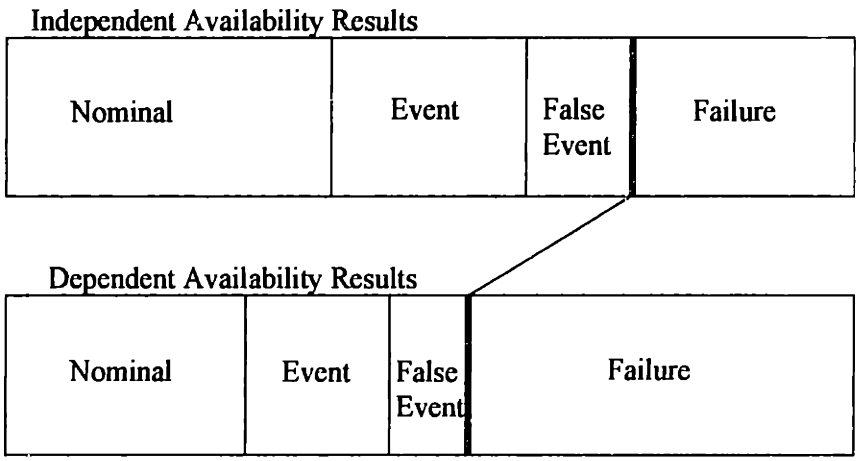


*Figure 3.15: Adjustment of Non-Failed States to Account for Dependent Availability*

In addition to naturally occurring events, some events occur whenever an event of another function occurs, as defined by a type 2 dependency described in section 3.3.4. It is assumed that the mean time to recovery for the event is the same no matter what caused it, so the probability of being in an event recovery state due to a type 2 dependency is given by Eq. 3.12 where mte1 is the mean time to event for the dependent function, mte2 is the mean time to event for the independent function, PE1 is the probability of being in an

event recovery state due to natural events, and PE2 is the probability of being in an event due to the independent function's events.

$$PE2 = PE1 \cdot \left(\frac{mte1}{mte2}\right) \tag{3.12}$$

Equation 3.12 simply states that the probability of being in an event recovery state due to each mode is inversely proportional to the mean time to event of that mode, given that the mean times to recovery are the same for both modes. This assumes that the probability of being in an event state is small. The number of operators is then used by substituting PE2 from Eq. 3.12 for the failure probability used in the equations of Appendix F to calculate H, and then substituting this into Eq. 3.13. The contribution from each function j with a type 2 dependency is added. Thus, Eq. 3.13 yields the number of operators required for event recovery caused by a type 2 dependency for a single function.

$$N_{E2} = \sum\left(\frac{LT}{OT}\right) \cdot H \cdot \left(\frac{A_d}{A_i}\right) \tag{3.13}$$

The probability of being in an event recovery due to the failure of another function (type 3 dependency) is calculated in the same manner as that due to another function's event. However, mte2 of Eq. 3.12 which represents the independent function's mean time to event is replaced by the overall mean time to failure for the independent function. The PE2 then is the probability of the dependent function experiencing an event due to a failure of the independent function. The procedures for the calculation of the number of operators is then identical to that of type 2 dependencies, and the contribution from each function k with a type 3 dependency is added as shown in Eq. 3.14.

$$N_{E3} = \sum\left(\frac{LT}{OT}\right) \cdot H \cdot \left(\frac{A_d}{A_i}\right) \tag{3.14}$$

The total number of operators required for a function's event recovery is then given by Eq. 3.15.

$$N_E = N_{EI} + N_{E2} + N_{E3} \tag{3.15}$$

In addition to calculating the operations requirements for failure and event recovery, the contribution due to normal, non-contingency operations must be included. Even when a function is fully operational, the level of automation may dictate some operator task load due to supervision, or other tasks which are dependent on the level of automation. This is entered into the model as a number of operators required continuously to perform such monitoring tasks, $N_S$.

# 4. Cost Models

## 4.1 Development Costs

In general, the development costs may be defined as encompassing the sub-costs shown in Figure 4.1. The cost breakdown is meant to illustrate the costs which are typically incurred at or before the operational stage of the program, and indicates the cost areas which are likely to be affected by automation. The model does not require that costs be broken down in such a manner, however, and any development cost model which incorporates the sub-costs of Figure 4.1 may be used.

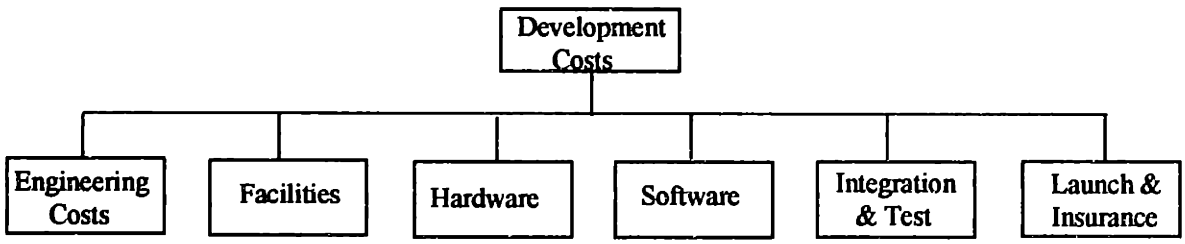


*Figure 4.1: Development Costs*

The sub-cost categories are self-explanatory. It is important to note however, that both ground segment and space segment costs must be included, as well as program-level costs. Also, the facilities costs included as development costs should only include the direct cost of buildings (construction or lease costs). Overhead costs such as administration and maintenance are included in the model as operations costs since they are incurred throughout the operational lifetime of the system.

Development costs are required to take place at the system beginning of life. If development costs are paid throughout the life cycle, this does not reclassify them as operations costs, but rather, the costs are entered into the model as a cash flow of monthly expenditures over the life of the system, and a discount rate is supplied to bring all future expenditures back to the present value for combination with other costs.

## 4.2 Operations Costs

The operations costs (Figure 4.2) are defined as those costs that are incurred over the operational lifetime of the system. Staff requirements can result from the basic system design, such as facilities maintenance costs, or may vary depending on the system's inherent availability and the level of automation. Operations costs are a function of the system availability since increased failures may result in an increased operations staff. The following sections refer to Figure 4.2, and describe the model inputs required for each operations sub-cost. These inputs are used in combination with the availability results to estimate both the staffing requirements and the operations cost.

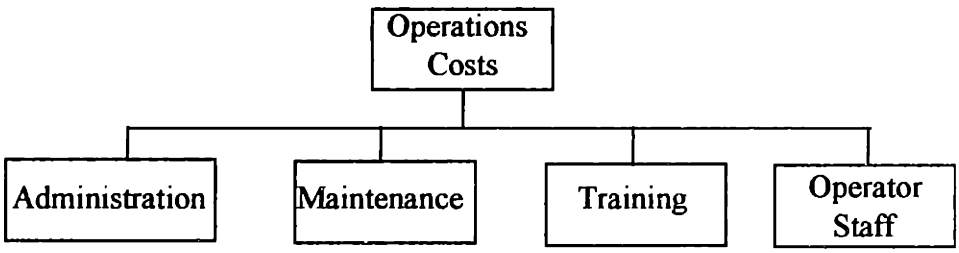


*Figure 4.2: Operations Costs*

### 4.2.1 Administration

Administration costs include all salaries to persons not responsible for training, direct operations, or maintenance. Such costs are typically grouped as overhead, and some organizations use rules of thumb for estimating overhead costs as a function of the operator staffing level. In such cases, non-operator personnel costs may be provided as a per-operator cost of support staff.

### 4.2.2 Maintenance Costs

Maintenance costs are defined mainly by the size and complexity of the ground facilities. These costs include all hardware and software maintenance for the system. Software maintenance may be required for the space segment as well, but this is usually in response to an event or failure of the spacecraft, and therefore would be the responsibility of the satellite operator. Such costs are counted separately since they depend on the probability of such events occurring. However, any planned upgrades should be included as part of maintenance costs.

### 4.2.3 Training Costs

Training of operators may result from operator turn-over, or from continuing training sessions to maintain a desired level of readiness for critical failures. As with non-operator costs, training costs may be strongly correlated with the operations staff, and therefore may be entered into the model on a per-operator basis.

### 4.2.4 Operations Staff Costs

Operations staff costs are determined based upon the skill level of the operator, which determines each operator's salary, and the number of operators required at each skill level. The calculation of the number of operators was described in section 3.5. The operator costs are calculated by multiplying the number of operators at each skill level by the salary paid to those operators. The sum of all operator salaries is the operator cost as shown by Eq. 4.1

$$OC_i = N_F \cdot S_F + N_E \cdot S_E + N_S \cdot S_S \tag{4.1}$$

where $OC_i$ is the operations cost for function i, $N_F$ is the number of operators required for function i's failure recoveries, $S_F$ is the salary paid to those operators, $N_E$ and $S_E$ correspond to event recoveries for function i, and $N_S$ and $S_S$ account for supervisory workloads for function i. The total operations cost is then calculated by the sum over all functions.

The simple sum of functional operations cost represents a total annual operations cost assuming that each operator works 24 hours per day, 365 days per week. To account for a 40 hour work week, the operations costs need to be multiplied by 4.2 (40 x 4.2 = number of hours in a week):

$$OC = \sum_i 4.2 \cdot OC_i \tag{4.2}$$

## 4.3 Opportunity Costs

In order to unambiguously compare systems, a metric combining the worth of availability and cost savings must be created. This metric, the life cycle cost, uses the concept of an opportunity cost to convert the system's availability to constant dollars so that a single life cycle cost may be calculated which accounts for both direct costs and availability issues. The relationship between the opportunity cost and each mission objective availability is subjective in nature, and should be used carefully. The following sections describe the general formulation of the opportunity cost function for two major classes of systems.

### 4.3.1 Commercial Systems

For commercial systems, mission objectives directly produce revenues. The opportunity costs arise from any lost revenue which results from the system's unavailability.

Most systems do not reach a state of full resource usage (saturation) immediately. There is often a period during which the number of subscribers is less than the maximum number the system is capable of supporting, often referred to as the system capacity. Therefore, even if the system is 100% available, there will be some loss of potential revenue due to the development of the market. This can be represented by a market factor, M which is the fraction of the total system capacity which is expected to be used at a given point in time (thus, $0 \leq M \leq 1$). As the time on orbit increases, the market factor may tend towards unity, indicating that the market demand is high enough that the system will saturate. $R_{max}$ represents the maximum revenue generated by the mission objective when $M = 1$. The expected system revenue, ER, is given by Eq. 4.3. Remember that M is a

function of time, so Eq. 4.3 can be used to calculate the expected revenue at any point in the entire system life.

$$ER(t) = M(t) \cdot R_{max} \tag{4.3}$$

The expected revenue defines a baseline from which opportunity costs may be calculated. Any decrease in the system availability will result in revenues less than those expected. The difference between the expected revenue and the actual revenue is the opportunity cost. There are two mechanisms which reduce revenues with decreasing availability of the mission objective. The most obvious is the direct loss of revenue which may result if customers are not charged during times for which the mission objective does not provide service to them. The other mechanism results from the fact that most commercial systems provide a service for which there is another provider.[†] This elasticity factor, $\alpha$, includes all losses due to the mission objective's unavailabilities, and can be combined with the expected revenue to define the actual revenue, AR. Thus, with the expected revenue defined as a function of time t, and the elasticity factor, $\alpha$, defined as a function of the availability, A(t), the actual revenue becomes a function of both parameters as given by Eq. 4.4.

$$AR(t) = ER(t) \cdot \alpha(A(t)) \tag{4.4}$$

The opportunity cost cash flow rate, OPC(t), is then shown in Eq. 4.5.

---

[†] In such cases, the availability of the system can be a factor in the determination of the market demand. As the mission objective becomes less available, more customers will go to other providers, or the price charged for the service may be reduced to maintain the desired level of system saturation. Thus, there is a second factor which is a function of the availability, therefore also a function of time, which accounts for the loss of revenues as the availability is reduced.

$$OPC(t) = ER(t) - AR(t) = \left(1 - \alpha(A(t))\right) \cdot ER(t) \tag{4.5}$$

The life cycle costs are then defined as the sum of development, operations, and opportunity costs as shown in Eq. 4.6, restated below.

$$LCC = DC + OC + OPC(t) \tag{4.6}$$

### 4.3.2 Scientific / Government Systems

For non-commercial systems, it may not be possible to define an opportunity cost in dollars. Subjective factors such as value of scientific data or the value of performance for a military system are difficult to measure. For scientific satellites, the worth of science may be estimated by dividing some measure of the desired scientific return by allowable cost of the system. The scientific return might be measured by area mapped for a remote mapping satellite, or by time of data collection for other missions. Utils may also be defined to measure the relative value of mission availability.

Assuming that a value of the mission objective can be defined, this can be used in place of $R_{max}$ in Eq. 4.5 with the market coefficient set to unity. For military systems, there may be no real opportunity cost, but rather a system availability which must be met to meet system requirements (this may also be true with other systems). It therefore may not make sense to try to define an opportunity cost, but rather to look at the system availability and cost as separate metrics.

## 4.4 Life Cycle Costs

Once the opportunity costs have been estimated as a function of time for each mission objective, they are combined with the development and operations cash flows to define a life cycle cost. In order to compare systems which may have different cash flows, the present worth value of the life cycle cost cash flow is calculated using a discount rate.[11] The discount rate for commercial systems would be set equal to the desired system rate of

return, and for other systems would be set depending on the interest rate applied to borrowed funds, or another appropriate rate.

For the case where no meaningful opportunity costs can be derived, the system availability and present worth value of just the development and operations costs would be used separately. The lowest cost system which meets the availability requirements would represent the optimal system by this method. Also, the incremental cost of attaining improved availability is presented clearly, and this may result in a different choice for the "optimal" level of automation.

# 5. SOCRATES

## 5.1 Overview

SOCRATES (Satellite Operations Cost and Reliability Analysis Toolkit for the Evaluation of Systems) is a software tool developed by MIT and the Charles Stark Draper Laboratory that links a graphical user interface with the modeling approach described in chapters 3 and 4. Through SOCRATES, the user is able to interactively define the system in detail, perform the functional decomposition, and specify the levels of automation for each function. Software performance and human reliability and performance estimates for event and failure recovery are entered into the model by the user. The user must also enter operations cost data, and development costs can be provided through one of several options. The functional interdependencies are also defined, as well as satellite, control center, and mission objective requirements through the use of a GUI (Graphical User Interface). Revenues are provided for each mission objective. The market and availability factors used to define opportunity costs are provided via a fifth order polynomial curve fit to the actual functions.

Figures 5.1 and 5.2 show the SOCRATES GUI. Figure 5.1 illustrates the functional decomposition for a typical satellite, and Figure 5.2 shows the level of automation definition for a space function. The level of automation is changed with slide buttons, and the labeled arrows in the diagram display the information flow for that particular level of automation. The GUI also allows objects such as satellites or functions to be saved independently. This allows new system definition to build upon earlier systems. Rather

than re-entering all of the data again, a saved satellite or series of saved functions may be loaded, and then modified as needed.

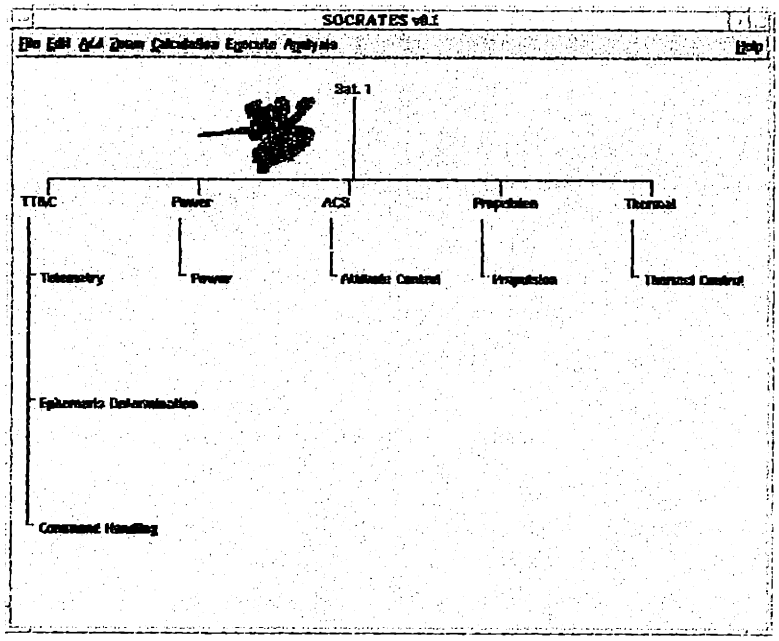


*Figure 5.1: SOCRATES Screen Showing Functional Decomposition*

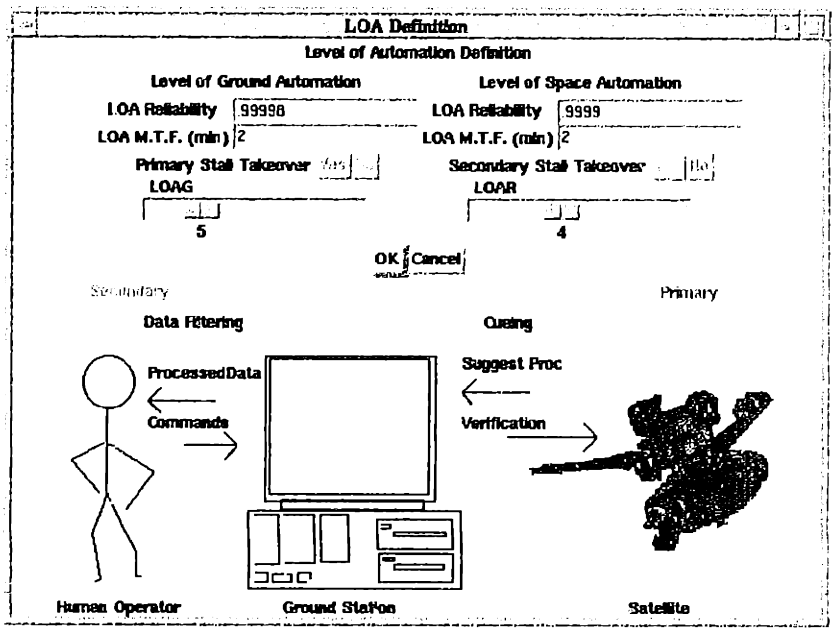


*Figure 5.2: SOCRATES Screen Showing Level of Automation Definition*

In addition to allowing the system variables to be entered, parameters governing the Markov calculations are also provided by the user. The time step may be varied to adapt to systems with different dynamics. The user is also able to specify output file names for cost and availability data which the model generates. These files provide cost and availability data in a comma delimited format to allow data viewing through many common commercial plotting packages. In addition, SOCRATES has a plotting tool which allows plotting of one or more variables as a function of time. Currently, the tool is implemented on a Sun workstation using C++ for the Markov engine, and tcl/tk to run the graphical user interface.

## 5.2 Data Flow

Figure 5.3 is a more detailed representation of Figure 2.3. The inputs and models have been divided into smaller elements. This has been done to more clearly describe the data requirements at each stage in the methodology. The letters labeling the arrows between blocks represent data flow. Table 5.1 lists the information represented by each letter. Further information regarding the algorithms and data requirements for SOCRATES can be found in the SOCRATES User Manual.[12] The details of this model are included in Appendix H.

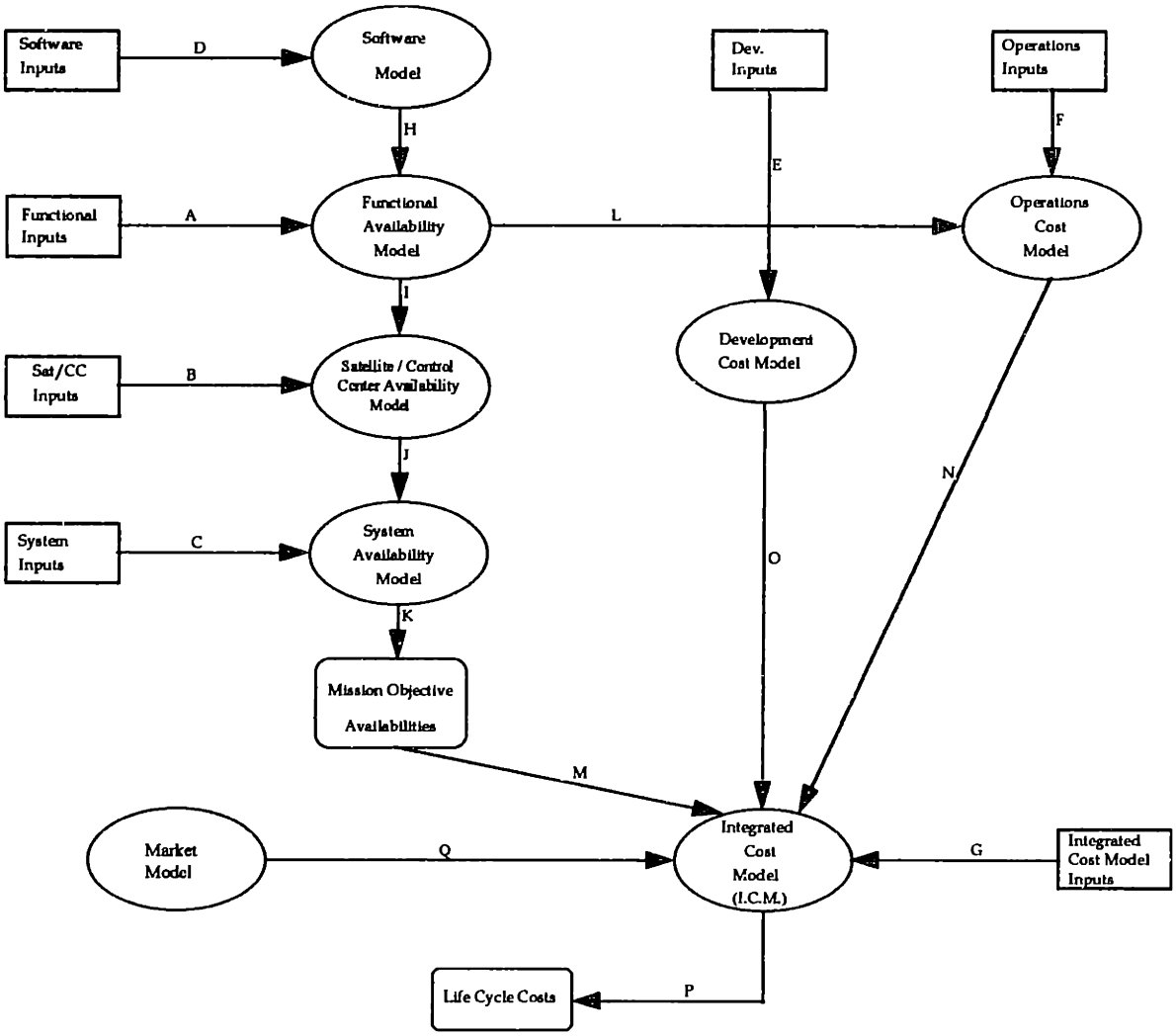


*Figure 5.3: SOCRATES Modeling Methodology*

Table 5.1: Model Data Flow

| Label | Data |
|---|---|
| A | Functional decomposition, failure rates, event rates, permanent failure rates, false event rates, levels of automation, human operator performance characteristics, computer processor performance, functional interdependencies. |
| B | Satellite and control center functional requirements. |
| C | Mission objective's requirements of satellite and control center objectives. |
| D | Software execution times, and reliability or test data. |
| E | Development cost inputs (cost drivers, cost breakdowns). |
| F | Operator salaries, non-operator related operations costs (admin., training, maintenance). |
| G | Mission objective opportunity costs as a function of availability, discount rate. |
| H | Software reliabilities and execution times. |
| I | Independent and dependent functional availabilities, steady state probability vectors. |
| J | Satellite and control center objective availabilities. |
| K | Mission objective availabilities |
| L | Functional steady state probabilities of being in a recovery state, functional interdependencies. |
| M | Mission objective availabilities |
| N | Development costs cash flow |
| O | Operations costs cash flow |
| P | Life cycle costs (cash flow and present worth value) |
| Q | Market Function (Market demand as a function of time) |

## 5.3 Flexibility

SOCRATES has been written in a way that facilitates customization. The GUI has been written separately from the actual computational engine. The interface between the two takes place through a set of data files. This allows independent customization of the GUI or engine. As long as each adheres to the same data file format, there are no other additional interface concerns.

Within the GUI, the development cost model allows a customer specific model to be chosen. The default external model is based upon the US Air Force's Unmanned Spacecraft Cost Model, Version 7.[13] The GUI's internal processing has been written in a separate file from the external development cost model. The file is called if the external model option is chosen within the main GUI, and returns a total system cost cash flow.

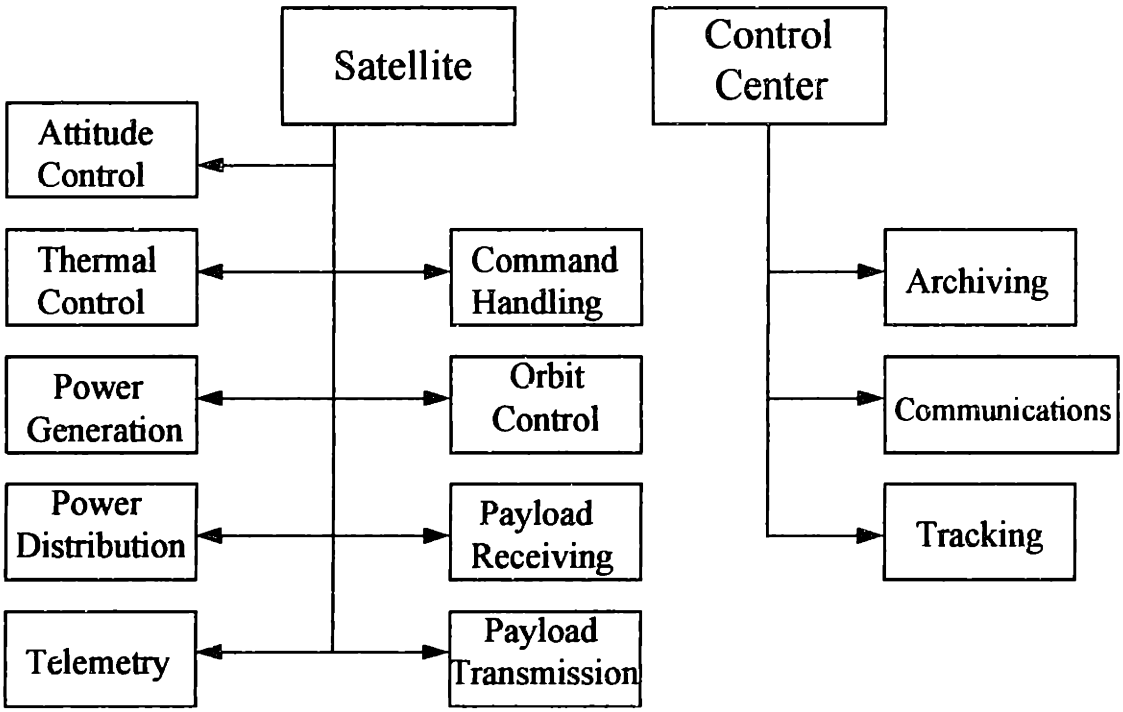


*Figure 6.1: Functional Block Diagram*

The trade to be studied involves the automation of the tracking function, which is a ground function in the baseline system definition. Tracking was chosen since it is a fairly simple function under current consideration for automation. The automation of satellite navigation has been demonstrated by Microcosm's MANS experiment[4] which suggests that such automation is both technically possible and useful in reducing operations costs.[5] It is assumed that this baseline system uses a radar tracking system at the control center to determine the spacecraft's position. Increasing the level of automation will involve allowing the ground computer to perform tasks normally done by the human operator. For higher levels of automation, the function itself will be moved from the control center to the satellite, and tracking will be performed on board through a GPS receiver.

Tables 6.1 and 6.2 list the baseline levels of automation for event and failure recovery for each of the 12 functions. The effects of the levels on the modeling inputs are discussed in detail in Appendix G-1.

*Table 6.1: Baseline Levels of Automation for Event Recovery*

| Function | Level of Remote Automation | Level of Ground Automation |
|---|---|---|
| Archiving | - | Paging |
| Tracking | - | Data Filtering |
| Control Center Comm. | - | Cueing |
| Attitude Control | Supervision | Data Filtering |
| Power Generation | Paging | No Automation |
| Power Distribution | Paging | Data Filtering |
| Telemetry | Fully Automated | - |
| Command Handling | Cueing | No Automation |
| Orbit Control | No Automation | Data Filtering |
| Payload Receive | No Automation | No Automation |
| Payload Transmit | No Automation | No Automation |
| Thermal Control | Paging | Cueing |

*Table 6.2: Baseline Levels of Automation for Failure Recovery*

| Function | Level of Remote Automation | Level of Ground Automation |
|---|---|---|
| Archiving | - | No Automation |
| Tracking | - | No Automation |
| Control Center Comm. | - | No Automation |
| Attitude Control | No Automation | No Automation |
| Power Generation | Paging | No Automation |
| Power Distribution | No Automation | No Automation |
| Telemetry | No Automation | No Automation |
| Command Handling | No Automation | No Automation |
| Orbit Control | No Automation | No Automation |
| Payload Receive | No Automation | No Automation |
| Payload Transmit | No Automation | No Automation |
| Thermal Control | No Automation | No Automation |

The transition rates (mean time to event, false event, failure and permanent failure) for the functions are given in Table 6.3. The event and false event rates are engineering estimates, but are not based upon any specific system. The failure rates are based upon an Air Force report on reliability prediction for spacecraft.[14] Functions for which a transition cannot occur have a mean time to transition set to infinity. In addition, the ground function permanent failures were assumed to never occur since equipment on the ground can always be replaced. The permanent failure rates were estimated based upon satellites from Project Renaissance (an MIT graduate space systems design course).[15] Note that the large permanent failure times result from high end of life reliabilities. For example, a 756 year mean time to permanent failure corresponds to a reliability of 0.987 after 10 years.

*Table 6.3: Transition Times for Single Satellite*

| Function | Mean Time to Event | Mean Time to False Event | Mean Time to Failure | Mean Time to Perm. Failure |
|---|---|---|---|---|
| Archiving | 4 min. | ∞ | 4 yrs. | ∞ |
| Tracking | 1 day | ∞ | 3 yrs. | ∞ |
| Control Center Comm. | 6 hrs. | ∞ | 10 mo. | ∞ |
| Attitude Control | 10 min. | 1 wk. | 6 yrs. | 195 yrs. |
| Power Generation | 30 min. | 1 wk. | 1 yr. | 328 yrs. |
| Power Distribution | 30 min. | 1 wk. | 4 yrs. | 328 yrs. |
| Telemetry | 4 min. | ∞ | 6 yrs. | 195 yrs. |
| Command Handling | 6 hrs. | ∞ | 7.5 yrs. | 251 yrs. |
| Orbit Control | 1 mo. | 1 yr. | 6 yrs. | 328 yrs. |
| Payload Receive | 1 mo. | ∞ | 1 mo. | 756 yrs. |
| Payload Transmit | 1 wk. | ∞ | 1 mo. | 756 yrs. |
| Thermal Control | 12 hrs. | 2 mo. | 4 yrs. | 1995 yrs. |

Table 6.4 shows the basic characteristics of the tracking function at each level of automation investigated. For the lower two levels of automation, a ground tracking station is the primary processor. For the next three levels, a GPS receiver on-board the satellite provides an estimate of the spacecraft ephemeris. The cases differ in the workload of each processor, and the ability to recovery from processor deficiencies.

There is an assumption here that the development cost for the radar station is linked to how often it is used. This is really only appropriate if the usage of the radar station is shared among several systems and the development cost is distributed based upon the time requirements of the systems. It is also appropriate if the usage of a radar station is bought as a service from the owner of the radar station.

*Table 6.4: Tracking Function Characteristics*

| Level of Automation | | | Responsibilities | | |
|---|---|---|---|---|---|
| Remote | Ground | Tracking Equipment | Human | Ground Computer | Satellite Computer |
| None | None | Radar Station | Set-up each pass manually | - | - |
| None | Data Filtering | Radar Station | Uses data from computer to set-up each pass | Filters Data | - |
| Cueing | Data Filtering | On-board GPS, Radar Station | Verification of satellite's estimate | Filters Data | Estimate of ephemeris |
| Paging | Data Filtering | On-board GPS, occasional Radar | Occasional Radar to determine ephemeris when required | Filters Data | Ephemeris determination, pages ground if unable |
| Fully Aut. | - | On-board GPS | - | - | Ephemeris determination |

Figure 6.2 shows the variation in the development cost with increasing automation. As the level of ground automation is increased from none to data filtering, the development cost rises mainly due to an increase in software requirements. There is a jump in development costs at cueing due to the requirement of a GPS receiver on board the satellite along with its accompanying software. However, the development cost then decreases with increasing automation. For a cueing level of automation, the radar equipment must be used as a check during each position check, and thus the development cost reflects both the usage of the radar station and the GPS receiver. Once the level of remote automation is increased to paging, the radar station is only needed in the event of a failure of the on board position determination system. Therefore, the radar station usage decreases considerably. The slight decrease in development cost at the full automation point is due to the lack of any paging software.

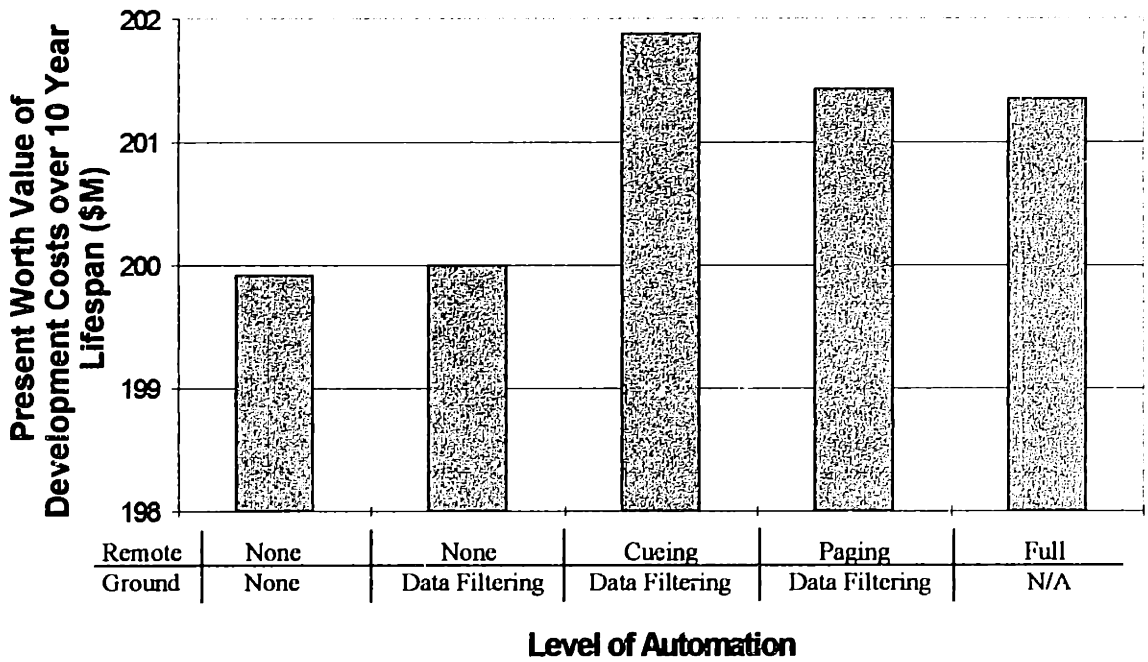


*Figure 6.2: Development Costs vs. Level of Automation (Single Satellite)*

Figure 6.3 shows the variation of operations costs for the levels of automation, and indicates that the function is sufficiently simple to allow high levels of automation. Once the human is eliminated from the loop to the extent that an operator is only needed when the flight code fails to correctly perform the function, the operations costs reach a constant value which reflects the total sum of operations costs due to the rest of the system. The variation of operations costs which results from software deficiencies is too small to be noticed on the plot, which is consistent with the assumption of a simple tracking function.

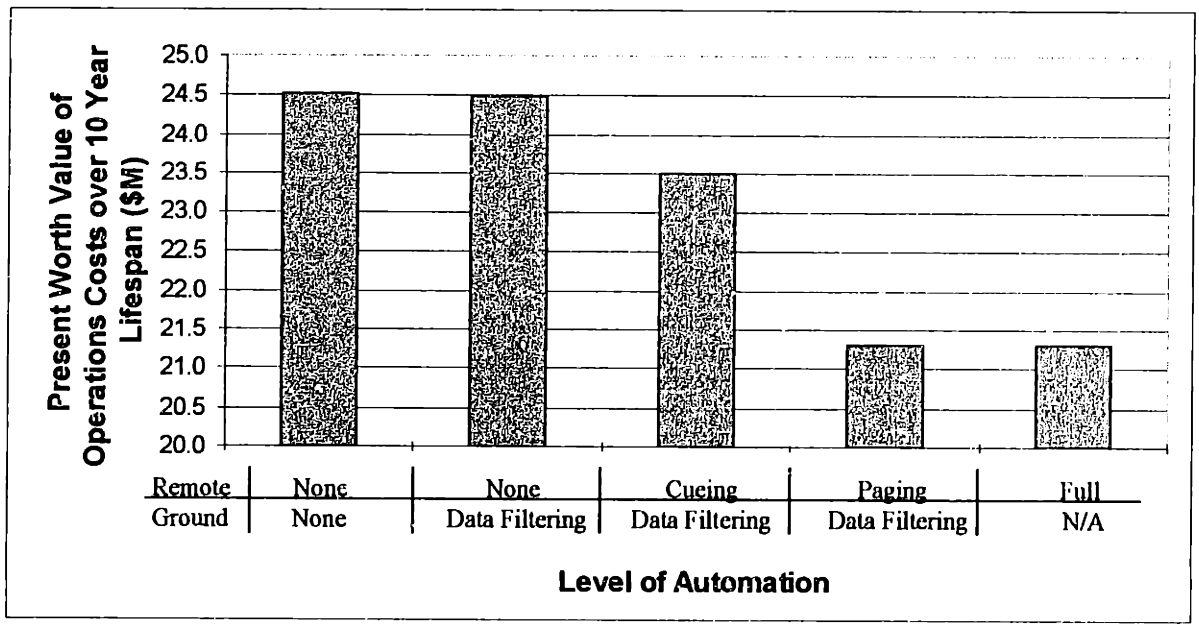


*Figure 6.3: Operations Costs vs. Level of Automation of the Tracking Function (Single Satellite)*

The opportunity costs do not vary with the level of automation. This is because tracking failures were assumed to occur very rarely. Therefore, the life cycle cost changes are due only to variations in development and operations costs. Operations costs were driven mainly by events rather than failures since events occurred once per day and failures were rare (once per 3 years).

Figure 6.4 shows the combined development, operations and opportunity costs brought to the present worth value for the levels of automation studied. It was assumed that the reliability of the human and automation were high for simple functions. In such cases, the system unavailability is dominated by the permanent failures rather than down time during operations. The life cycle cost curve shows that the decrease in operations costs are high enough for a highly automated system to justify the increased development costs. Thus, the optimal level of automation for the tracking function is to be fully automated.

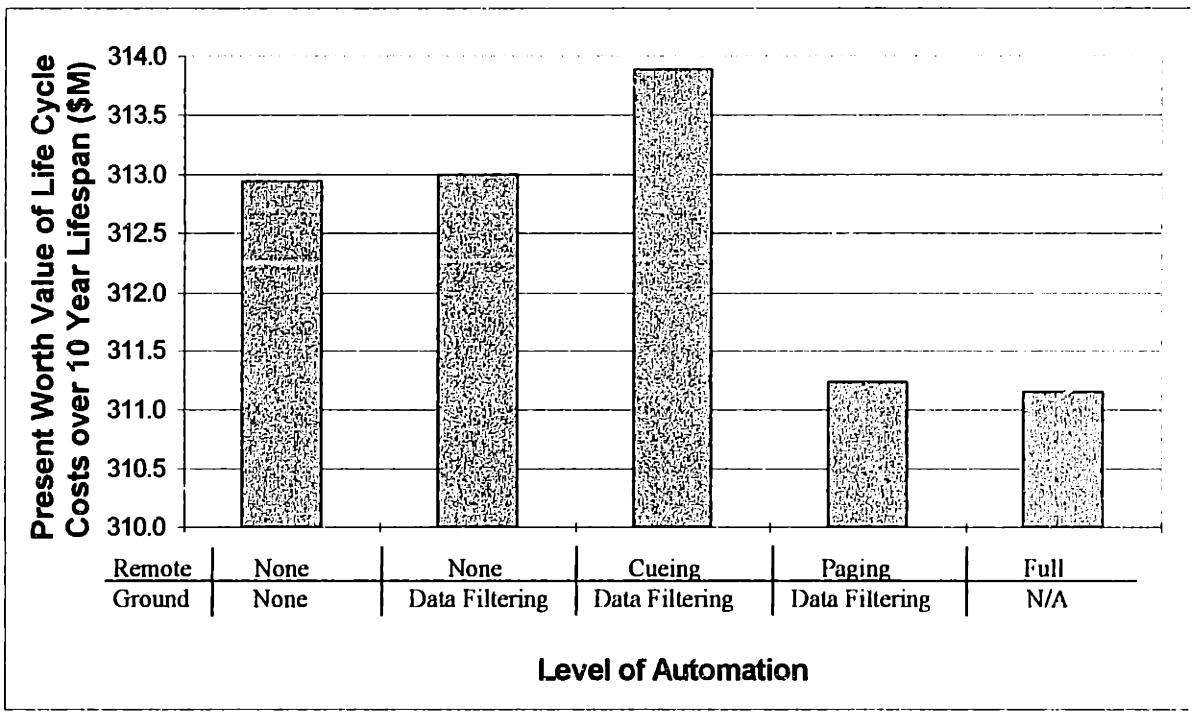


*Figure 6.4: Life Cycle Costs vs. the Level of Automation of the Tracking Function (Single Satellite)*

The curves in Figure 6.5 have been normalized by the baseline system costs to show relative changes. Automating the tracking function does not greatly affect the opportunity costs or development costs, but produces a 13% savings in operations costs. However, since operations costs make up only a small portion of the life cycle costs, the relative changes in life cycle costs are only a few percent.

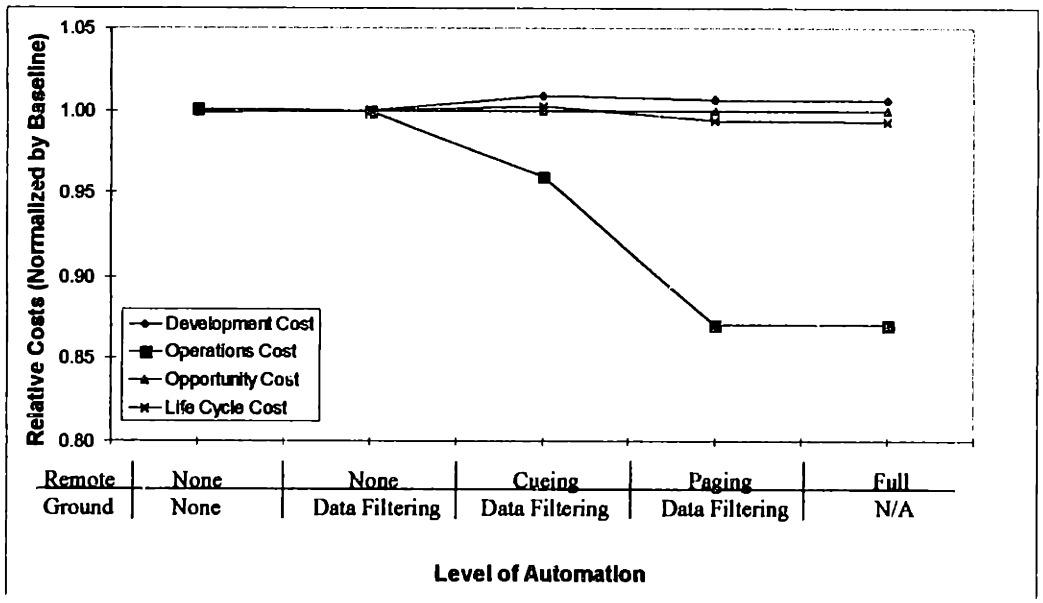


*Figure 6.5: Cost Curves Normalized by the Baseline System {No Remote Automation, Ground Data Filtering, Single Satellite}*

Remember that this case study only concerned automating one of 12 system functions. Therefore, the small changes in costs are not surprising. Applying the methodology to all system functions might greatly increase the effect on the total costs.

### 6.1.1 Sensitivity to Development Cost Inputs

The results of Figure 6.3 hold only for the assumed inputs. Since the tracking function was assumed to be relatively unimportant in meeting the mission objective, the opportunity costs do not vary significantly with the level of automation for this function. However, automation of the tracking function does impact both the development and operations cost. Since the development and operations cost inputs may be estimated, it is of interest to determine the sensitivity of the results to the inputs so that the validity of the results can be determined.

Based upon the general trend shown in Figure 6.3, it is expected that if the development costs related to tracking automation are sufficiently high, the optimal level of automation will shift from Fully Automated to one of the lower levels. Figure 6.6 shows the system

development costs which result from multiplying the tracking automation costs by a factor of three. This results in the adjusted life cycle cost curve shown in Figure 6.7. As can be seen, the increased development cost (which is characteristic of a more complex function) are high enough to outweigh any savings from operations costs which result as the level of automation is increased. Thus, the optimal level of automation is no automation. The break-even point for the development costs where the life cycle costs for no automation and full automation are equal occurs at 2.3 times the baseline development costs.

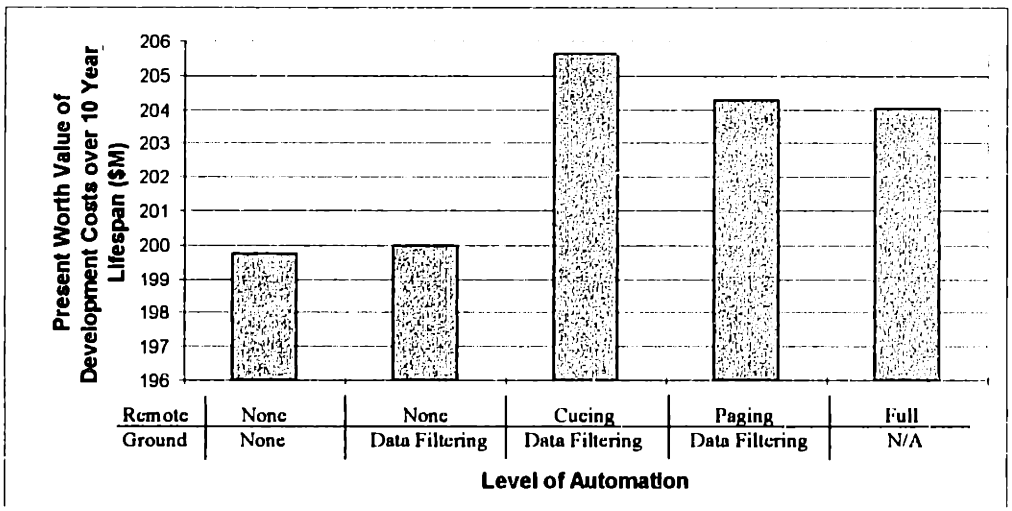


*Figure 6.6: Development Cost with 3x Increase in Automation Costs Relative to Baseline*

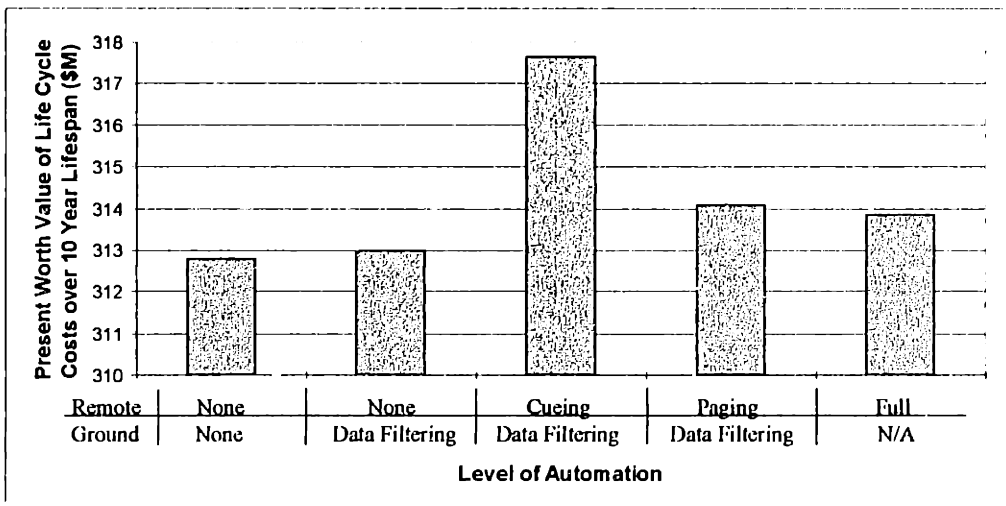


*Figure 6.7: Life Cycle Costs with 3x Baseline Automation Costs*

### 6.1.2 Sensitivity to Operations Cost Inputs

As an alternative to varying the development cost inputs, the operations costs may also be varied. Figure 6.8 shows the updated operations cost curves. It has been assumed that the operations costs for the tracking function are a third of the original estimate. Not surprisingly, decreasing the operations cost by a third has a similar effect as increasing the development cost by a factor of three as shown in Figure 6.9, except that the overall life cycle costs have been reduced. Thus, again it is more effective to use no automation. Note that the original development cost estimate has been used in this case. The break-even point for the operations costs where the life cycle costs for no automation and full automation are equal occurs at 0.43 times the baseline operations costs.

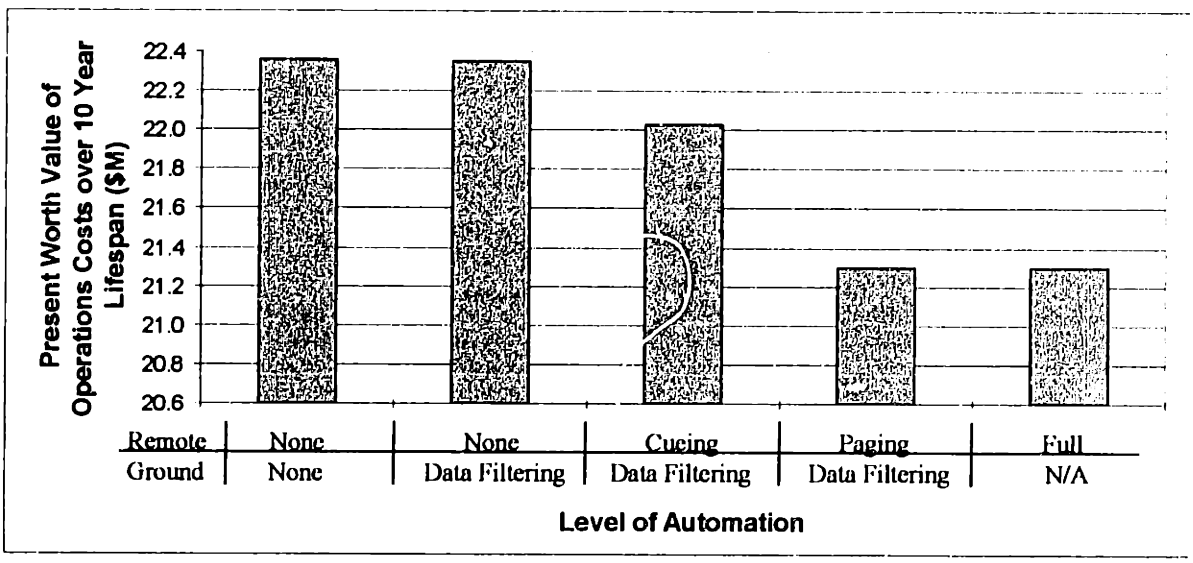


*Figure 6.8: Operations Costs 1/3 that of Baseline*

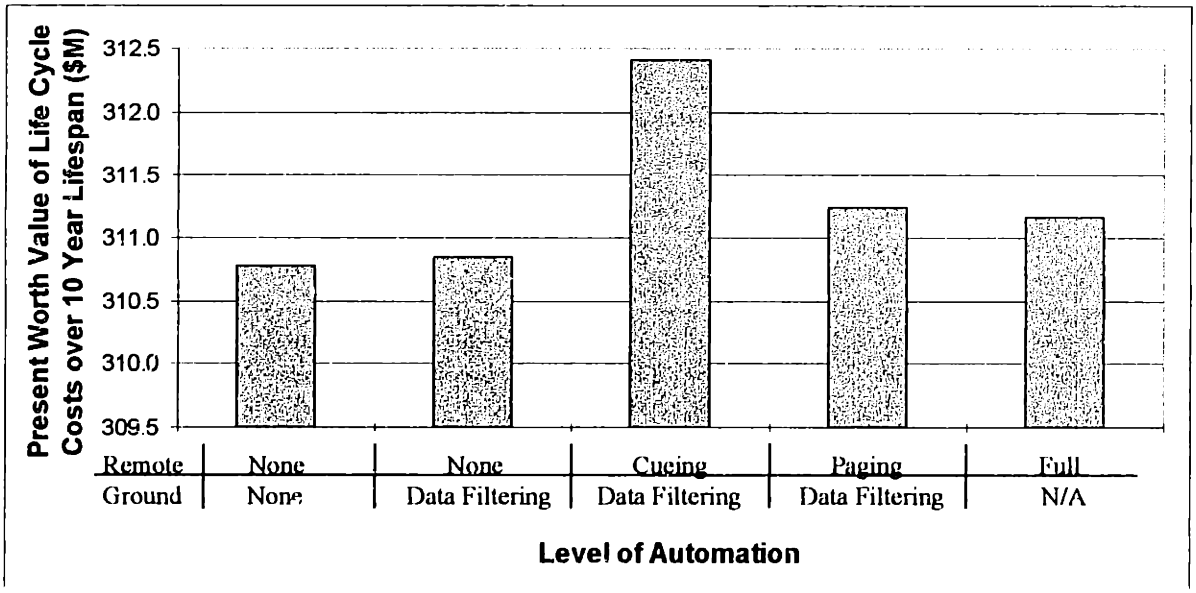


*Figure 6.9: Life Cycle Costs with 1/3 Baseline Operations Costs*

## 6.2 Automation Applied to Constellations of Commercial Communications Satellites

Although applying automation to a single satellite may be useful, one of the main drivers for the development of the methodology is the high operations cost which results from large constellations of satellites. This example shall use the methodology to determine how the automation trade changes as a function of the number of satellites in a constellation. The constellation is assumed to be constructed of identical communications satellites described in section 6.1. The input parameters are thus the same and are provided in Appendix G-1. The automation is applied to the two payload functions (Payload Transmit and Payload Receive) of Figure 6.1. The inputs regarding the automation of the payload functions are included in Appendix G-2. It is assumed that the payload functions, unlike the tracking function, are complex enough that automation will require a significant increase in development costs. Under this assumption, high levels of automation are not expected to be feasible for a single satellite.

It is assumed that a 90% learning curve was applied to the system. The learning curve is defined by Eq. 6.1,

$$TPC = C_{FU} \cdot L \tag{6.1}$$

where TPC is the total production cost, $C_{FU}$ is the first unit cost, and L is defined by,

$$L = N^{B}, \qquad B = 1 - \frac{\ln\left[\left(\frac{100\%}{S}\right)\right]}{\ln 2} \tag{6.2}$$

where N is the number of units (satellites), and S is the learning curve (90% here). The learning curve accounts for the economies of scale which result from the production of multiple "identical" units. Figure 6.10 shows the total production cost as the number of units increases for a 90% learning curve, as a fraction of the first unit cost. At 2 units, the total cost is nearly twice the first unit cost (actually 1.8). By 20 units, the total cost is just under 13 times the first unit cost.

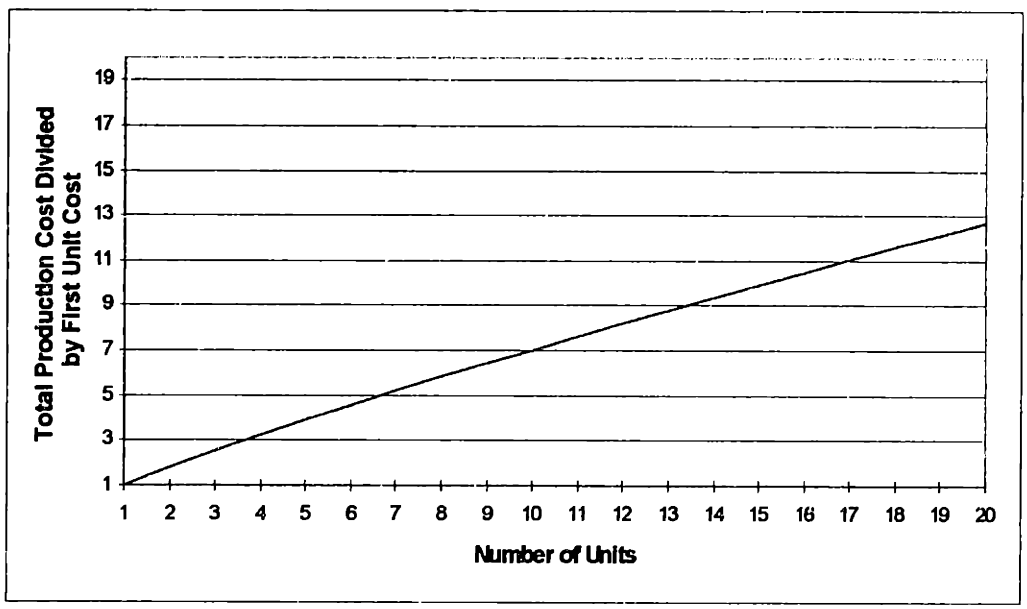


*Figure 6.10: Total Production Costs as Fraction of First Unit Cost vs. the Number of Units Produced (90% Learning Curve)*

The development costs per unit decrease with increasing size of the constellation. As the system grows, it is assumed that the operations costs and opportunity costs scale linearly with the constellation size while the development costs lag a linear relationship due to the learning curve effect. At some point, the savings in operations costs, multiplied by the number of satellites, outweighs the increase in development costs, and higher levels of automation may become optimal for that constellation size.

Figure 6.11 shows the results for the single satellite case. It is somewhat similar to the plot for the tracking automation trade (Figure 6.4), but the opportunity costs are more accentuated, and the lower levels of automation provide the lowest life cycle cost. The effect is that high levels of automation, which for the complex payload operations result in a decreased availability, are more "expensive" in a life cycle cost sense. Also, for the tracking trade, the increased development costs associated with data filtering were not offset by any operations or opportunity cost benefit. With the payload functions, however, the increased speed in which a human can detect and repair errors with data filtering results in decreased down-time once a failure occurs, which results in a decrease in opportunity costs. Since revenues are much more dependent on the payload than tracking, this opportunity cost savings is high enough to justify data filtering.

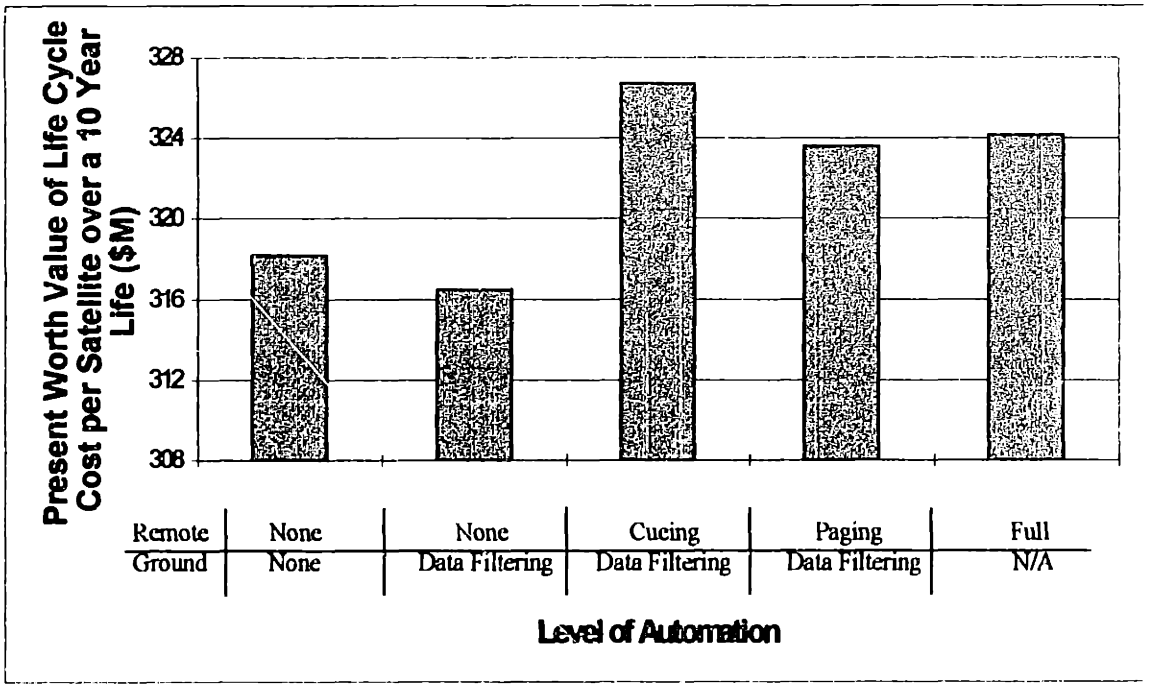


*Figure 6.11: Life Cycle Costs per Satellite vs. Level of Automation (1 Satellite)*

The results of the single satellite case hold for small constellations. However, the difference in the life cycle costs between the data filtering level of automation and the higher levels of automation decreases steadily as shown in Figure 6.12, which is for a 10 satellite constellation. Comparison of the per satellite life cycle cost with Figure 6.11 clearly illustrates the economies of scale which affect the development costs.

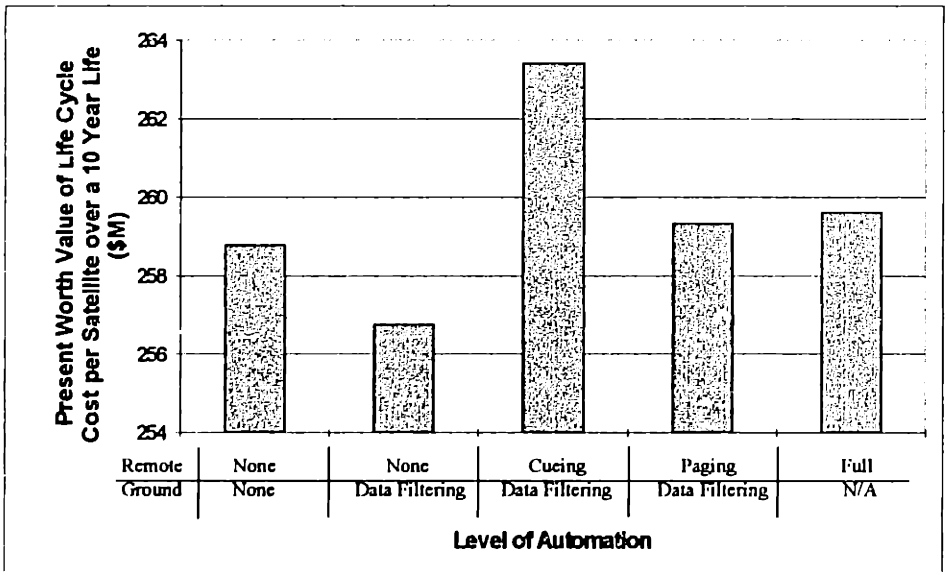


*Figure 6.12: Life Cycle Costs per Satellite vs. Level of Automation (10 Satellite Constellation)*

Although the data filtering level of automation remains the local optimum, the high levels of automation become less expensive than the no automation case for a constellation of 15 satellites as shown in Figure 6.13. This is also true for much larger constellations.

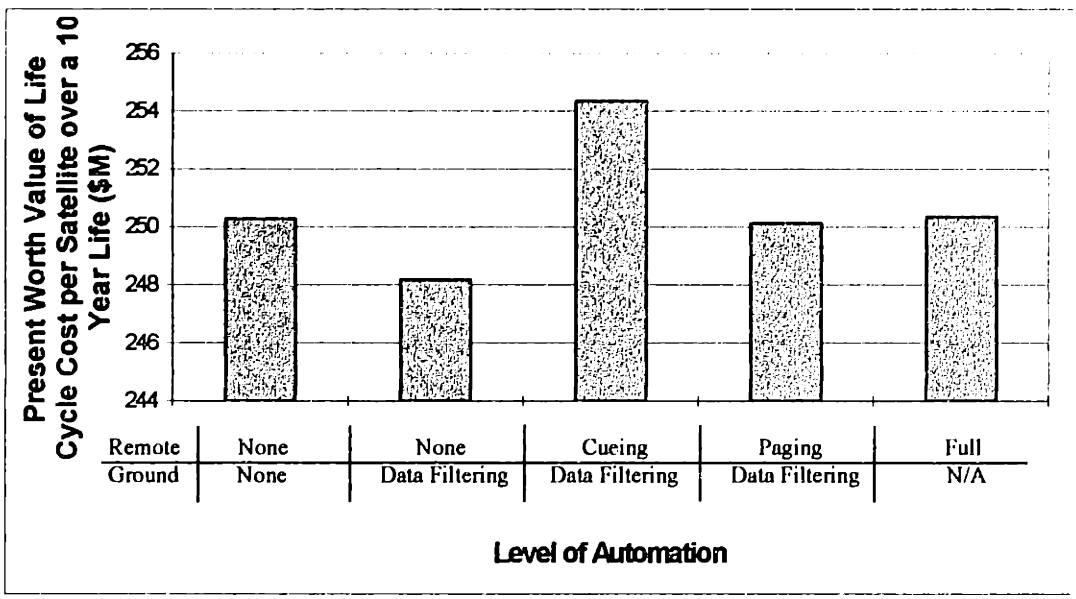


*Figure 6.13: Life Cycle Costs per Satellite vs. Level of Automation (15 Satellite Constellation)*

At a constellation size of 60 satellites (Figure 6.14), the benefits of decreased operations cost overcomes the higher development costs for the higher levels of automation. Even so, the fully automated case has a higher life cycle cost than the paging case due to the decreased system availability since processor deficiencies cannot be recovered from. The trend does not stop at 60 satellites. Even larger constellations would result in a larger benefit to automation relative to the data filtering case.

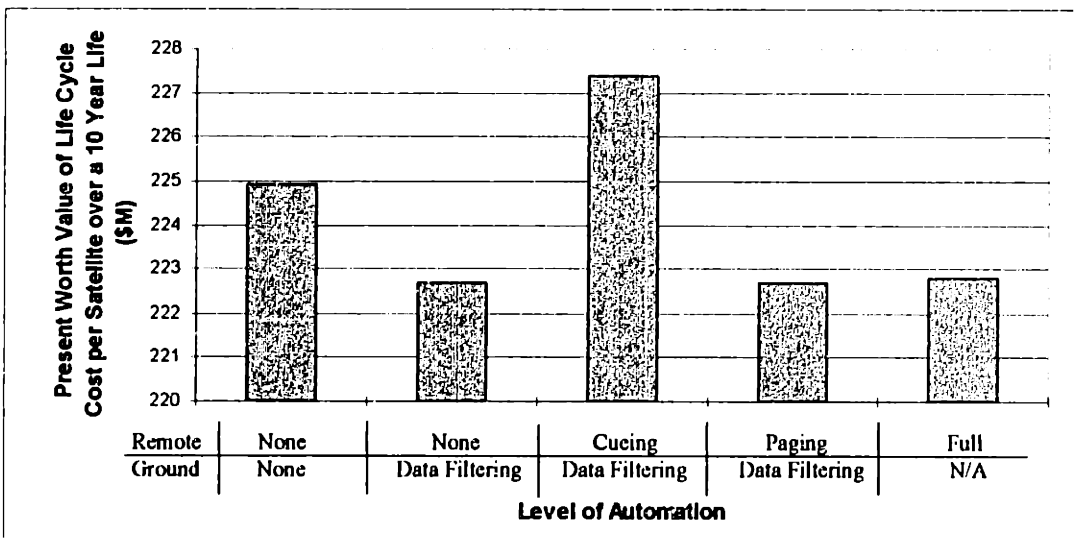


*Figure 6.14: Life Cycle Costs per Satellite vs. Level of Automation (60 Satellite Constellation)*

Figure 6.15 shows the life cycle cost per satellite as a function of the constellation size. The curves are normalized by the ground data filtering level of automation. Figure 6.15 clearly illustrates the effect of the learning curve. The cost per satellite for systems with automation gradually approach that of the un-automated system until, at 60 satellites, the system with paging has the lowest life cycle cost per satellite.

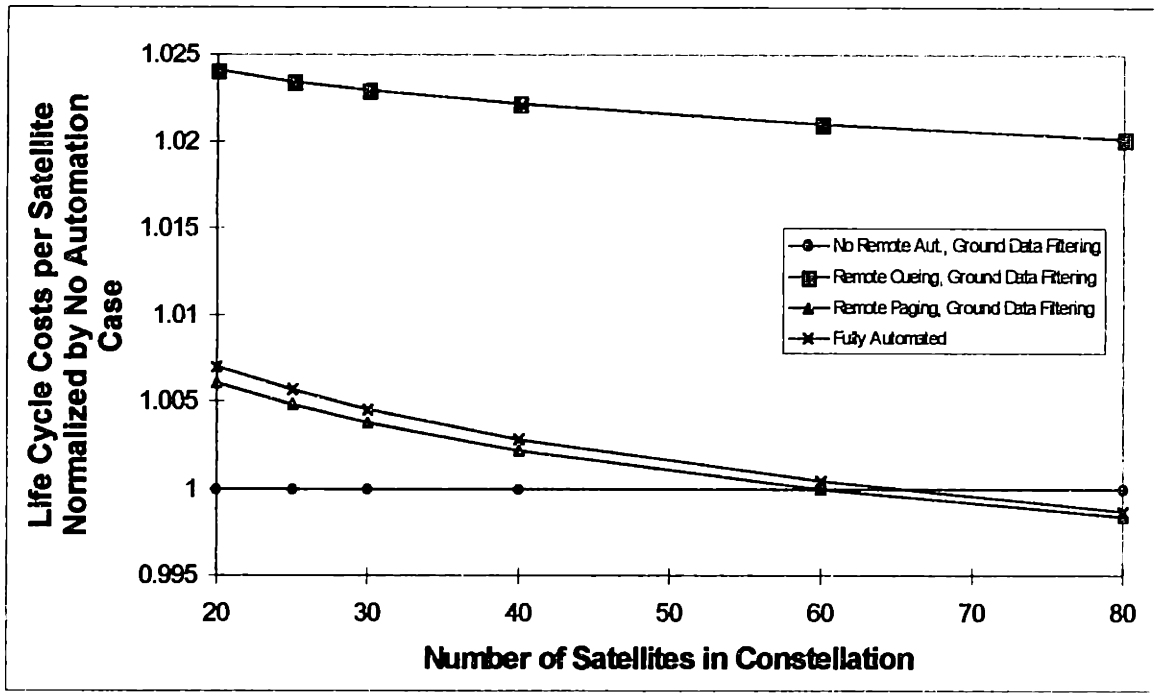


*Figure 6.15: Life Cycle Cost vs. Level of Automation of the Payload Functions for Constellations of Varying Size*

## 6.3 Automation for a Deep Space Mission

The above case studies showed how automation can reduce life cycle costs primarily through a reduction in operations costs. However, automation is often applied to remote systems because the delay times associated with the remote operations are large enough to make control of the spacecraft difficult or impossible. This is especially true for functions which require operator intervention often, and require that the recovery take place within a short period of time. Automation may also be used to increase flexibility, allowing the spacecraft to adapt to the changing environment and unforeseen anomalies. Such automation has been tested during the Clementine mission in 1994.[16]

This case study investigates the effect of time delay on a two-function deep space probe. The functions corresponding to the bus are assumed to be lumped into one function, and those corresponding to the payload are lumped into another. The functional block diagram is shown in Figure 6.16. Automation is applied to each function separately to determine how the optimal level of automation changes as the distance from Earth

increases. The four missions are: solar, Mars, Saturn, and a geostationary mission to form a basis of comparison. The delay times for each mission are shown in Table 6.5. Note that for the planetary missions, the worst case delay is assumed where the planet is opposite the Earth relative to the Sun.

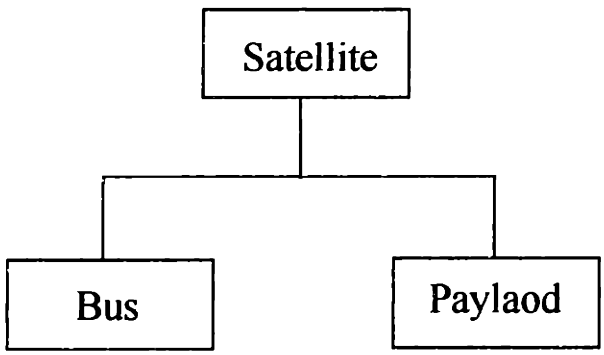


*Figure 6.16: Deep Space Satellite Functional Block Diagram*

*Table 6.5: Range and Round Trip Communications Times for Various Destinations within the Solar System*

| Destination | Maximum Range From Earth (meters) | Round-Trip Communications Time (min.) |
|---|---|---|
| Geosynchronous Orbit | $3.58 \times 10^{7}$ | 0.004 |
| Sun | $1.49 \times 10^{11}$ | 16.6 |
| Mars | $3.78 \times 10^{11}$ | 42 |
| Saturn | $1.58 \times 10^{12}$ | 176 |

The delay times are added to the recovery time once a failure or event has occurred. For human operators performing the recovery, the delay time is added to the total elapsed time, but the logged time remains the same. That is, a failure requires the same operator workload regardless of the round trip communications time, but the time to recovery for the function increases according to Table 6.5.

Rather than defining an opportunity cost, the life cycle cost was assumed to be the sum of development and operations costs alone. The availability was used to determine an expected value for the number of hours of data which would be collected over a 2 year

mission span. The metric used to determine the optimum was the life cycle cost divided by the expected hours of data collection (cost per hour data collection).

The missions were assumed to all have identical development costs (for the same levels of automation) regardless of the target. This cost was baselined at $150 million which is consistent with a NASA New Millennium mission.[17]

### 6.3.1 Bus Automation

Automation was first introduced to the bus function while holding the payload automation constant. The bus was assumed to be a simple function with failures occurring on the order of twice per month. Other inputs are contained in Appendix G.3.

Figure 6.17 shows the life cycle cost (development + operations) as a function of the level of bus automation for each mission. The life cycle costs follow similar patterns as shown in the previous case studies. Because of the above assumptions, the cost is not dependent on the mission (and therefore delay time). The operations costs are the same regardless of the mission because the delay time was not assumed to affect the failure rates. Therefore, the operations requirements remain unchanged regardless of the mission.

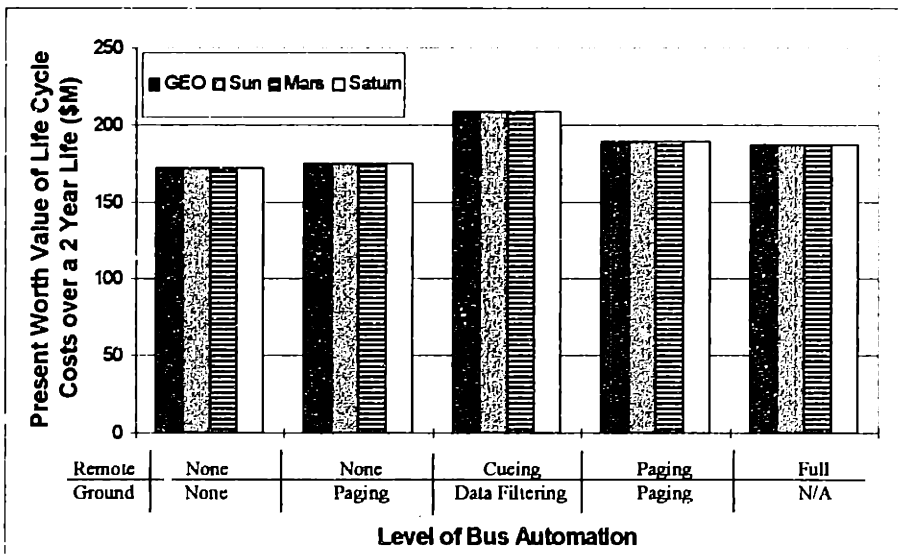


*Figure 6.17: Life Cycle Costs vs. Bus Level of Automation*

The increased delay time does, however, affect the bus function availability. The availability integrated over time results in an expected time of data collection. Figure 6.18 shows this as a percentage of the total possible data collection over the two year mission life. The effects of the increasing delay time with mission duration are very clear, resulting in increased down-time due to the longer elapsed times for failure recovery. However, the data yield does not vary much with the level of automation except for the fully automated case. This results from the assumptions that the bus function is simple, and failures are relatively rare. Thus, the trends of the Tracking automation case discussed in section 6.1 are again seen. The availability is not greatly affected by the level of automation if the function is simple, and if failures do not occur often. However, the fully automated case shows a decrease in the data yield. Since the bus function is fully automated, any failure which the automation cannot fix results in a permanent loss of the bus. Thus, the fully automated case has a slightly lower mean time to permanent failure than the other cases, which results in a decreased data yield.

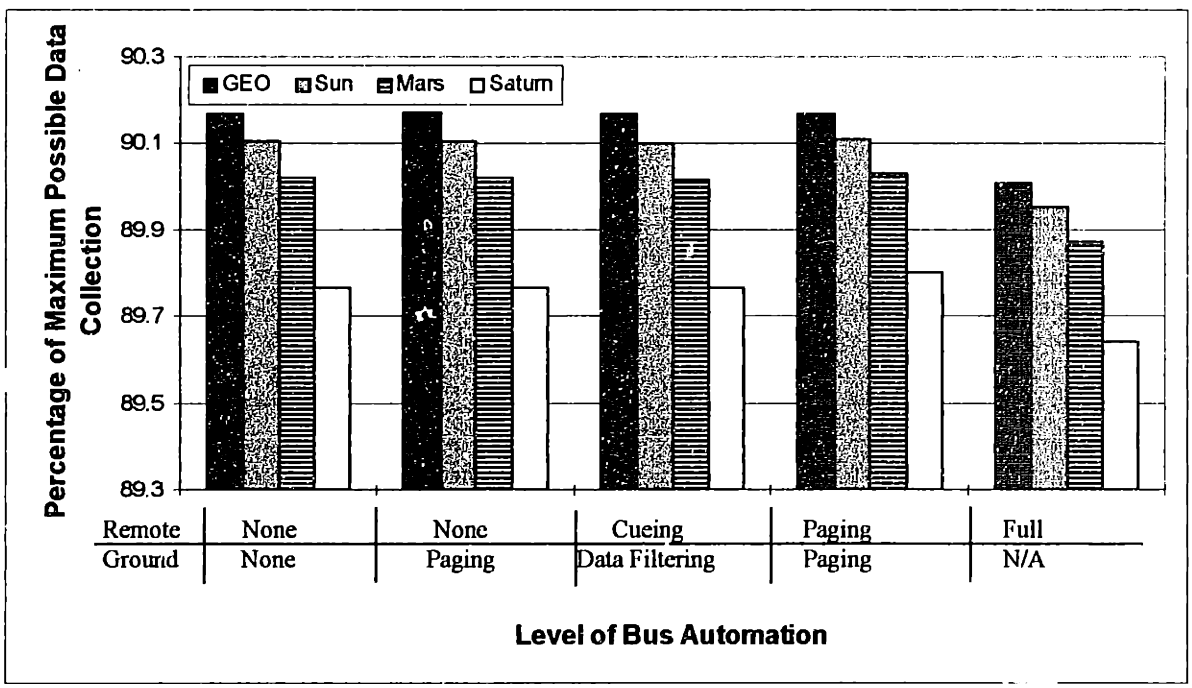


*Figure 6.18: Percentage of Maximum Possible Hours of Data Collection vs. Level of Bus Automation*

It should be noted that even though the data yield is affected by the mission, the changes are only on the order of 0.5%. This is a result of a low failure rate. Since failures do not occur often, the increased delay times do not have many opportunities to reduce the time of data collection.

The cost per hour of data collection is shown in Figure 6.19. Since the life cycle costs are identical for each mission, and the data yield varies by approximately one percent, it is not surprising that the cost per hour metric does not vary much from one mission to another. Note that the slight decrease in the data yield for the fully automated case (Figure 6.18) was not enough to outweigh the decreased life cycle cost at this level (Figure 6.17).

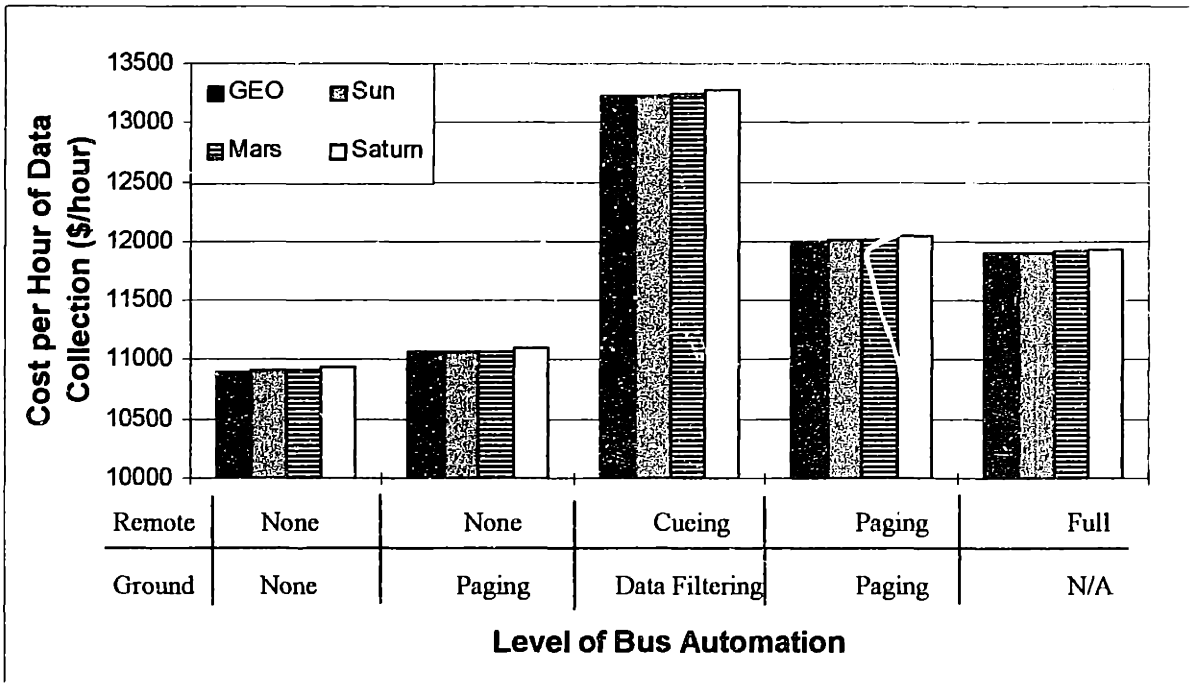


*Figure 6.19: Cost per Hour of Data Collection vs. Level of Bus Automation*

For the bus function, use of no automation minimizes the cost per hour data collection metric. This is due to the assumption that bus failures only occur twice per month. Therefore, the operations savings are too small compared to the development costs to justify automation. This low failure rate also led to a small impact in the data yield. This

holds for all missions because the 15 day mean failure time is large compared to the delay time ( $\leq$ 176 minutes).

### 6.3.2 Payload Automation

The payload function was automated assuming the optimal level of bus automation (i.e., no bus automation). Again, the development costs were assumed only to be a function of the level of automation and not the mission. However, it was assumed that the payload function was more complex than the bus function, and failures occurred more often (once every 4 hours). Remember that the definition of a failure is any occurrence that results in the unavailability of the function. It is assumed that for this case study, instrument scheduling is dynamic and requires updating every four hours. It is expected that this increased operations requirement will have a larger impact on the availability of the system, and therefore the data yield. Other input parameters are given in Appendix G-3.

The life cycle costs are shown in Figure 6.20. As with the bus automation, the payload life cycle costs are only a function of the level of automation. However, since the payload failure recovery was assumed to be more complex, the development cost increases with automation are higher than those for the bus.

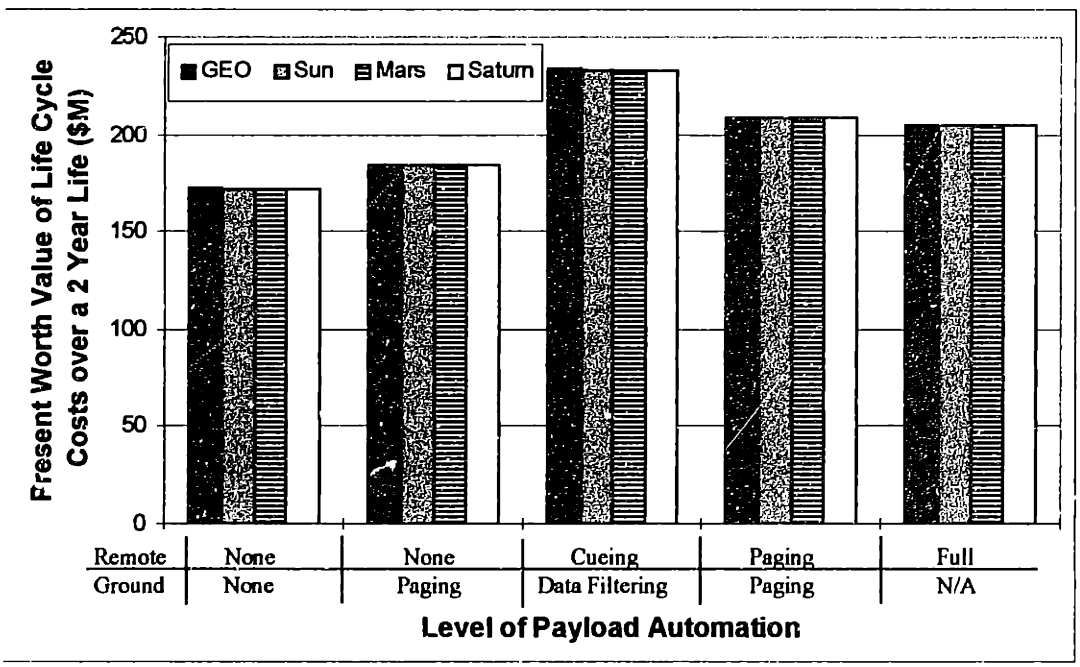


*Figure 6.20: Life Cycle Costs vs. Level of Payload Automation*

The percentage of maximum possible hours of data collection is shown in Figure 6.21. For the lower three levels of automation, the data yield is both a function of the level of automation and the mission. This is because the payload failures occur often enough to make the delay time significant. The increased speed of the ground computer over the human is also shown as an increased data yield for the ground paging case over the no automation case. The re-introduction of the human in the loop during all operations for the cueing case results in a decreased recovery time from the ground paging case. For the last two levels of automation in Figure 6.21, the satellite is the primary processor. The data yield is increased, but seems to be independent of the mission.

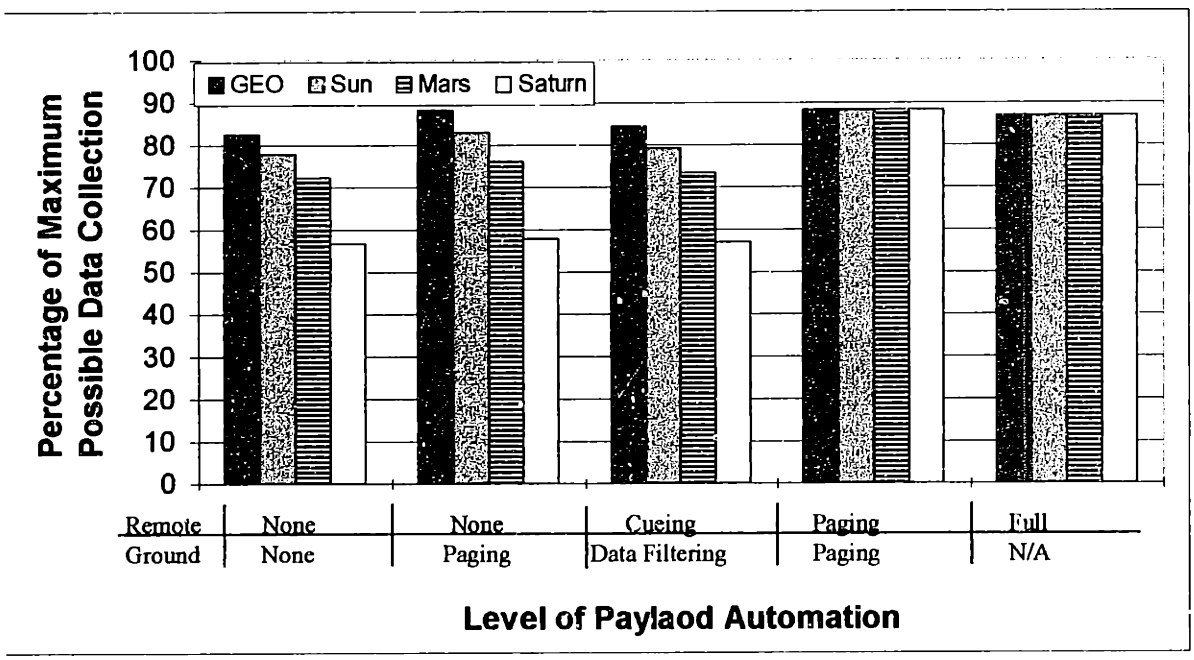


*Figure 6.21: Percentage of Maximum Possible Data Collection vs. Level of Payload Automation*

The cost per hour of data collection is shown in Figure 6.22. Here, the effects of the large payload failure rate are shown. Although the general trend for the GEO case remains the same as for the bus automation, this is not so for the farther missions. In fact, even for GEO, the ground paging case results in the minimization of the cost per hour of data collection. This is because the payload function's failure rate is high enough to make

operations costs a significant portion of the life cycle costs. Thus, the savings with automation are large enough to balance the increased development costs. However, since ground software is less expensive than flight software, it is more expensive for the GEO case to further automate.

The solar mission case also is optimized for ground paging. The increased delay time, and therefore decreased data yield is not yet enough to outweigh the difference in software costs between the ground and space. However, for the Mars and Saturn missions, the delay times are great enough to justify the higher cost in flight code, and the remote paging case results in the minimum cost per hour of data collection.

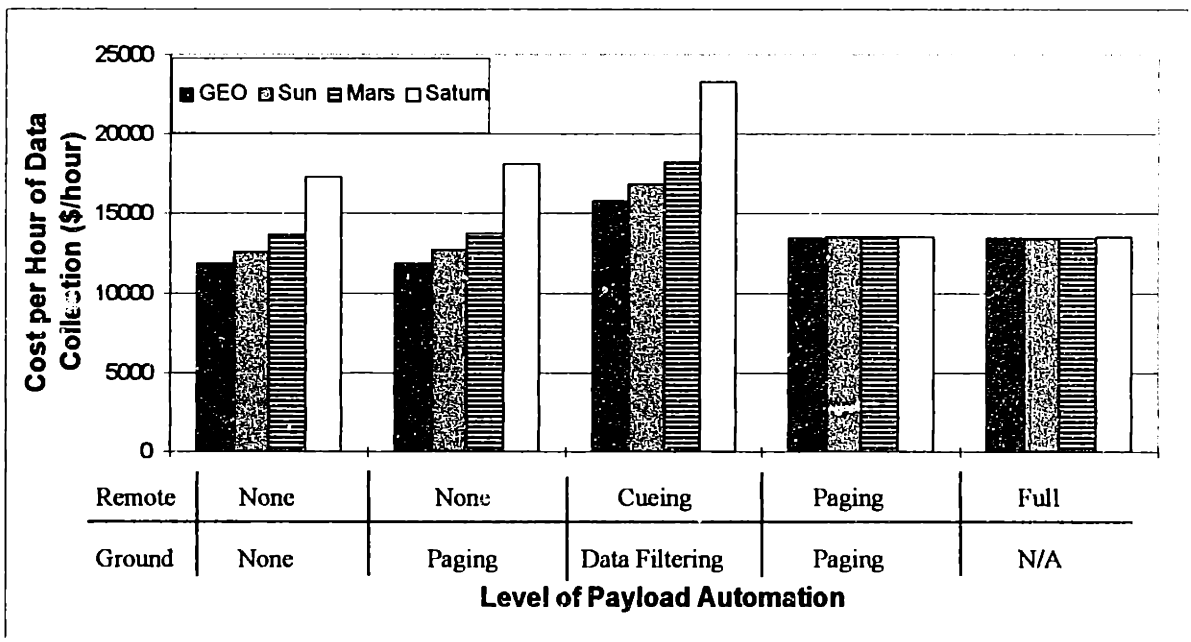


*Figure 6.22: Cost per Hour of Data Collection vs. Level of Payload Automation*

Figure 6.23 shows the dependence of the cost per hour metric on the round-trip communications time for low automation (ground processing) and high automation (space processing) of the payload function. The break-even point is clearly evident at just under 40 minutes. The curve also shows, as expected, that the performance of a highly automated system is not dependent on the delay time, but is for a system with a low level of automation. The cost per hour is nearly linearly dependent on the delay time, especially

for short delay times relative to the mean time to failure. This relationship is driven (in this case) by the availability. The deep space mission case study shows that for functions requiring frequent operations, automation will be required as the range (and therefore delay time) is increased in order to minimize the cost per data yield.

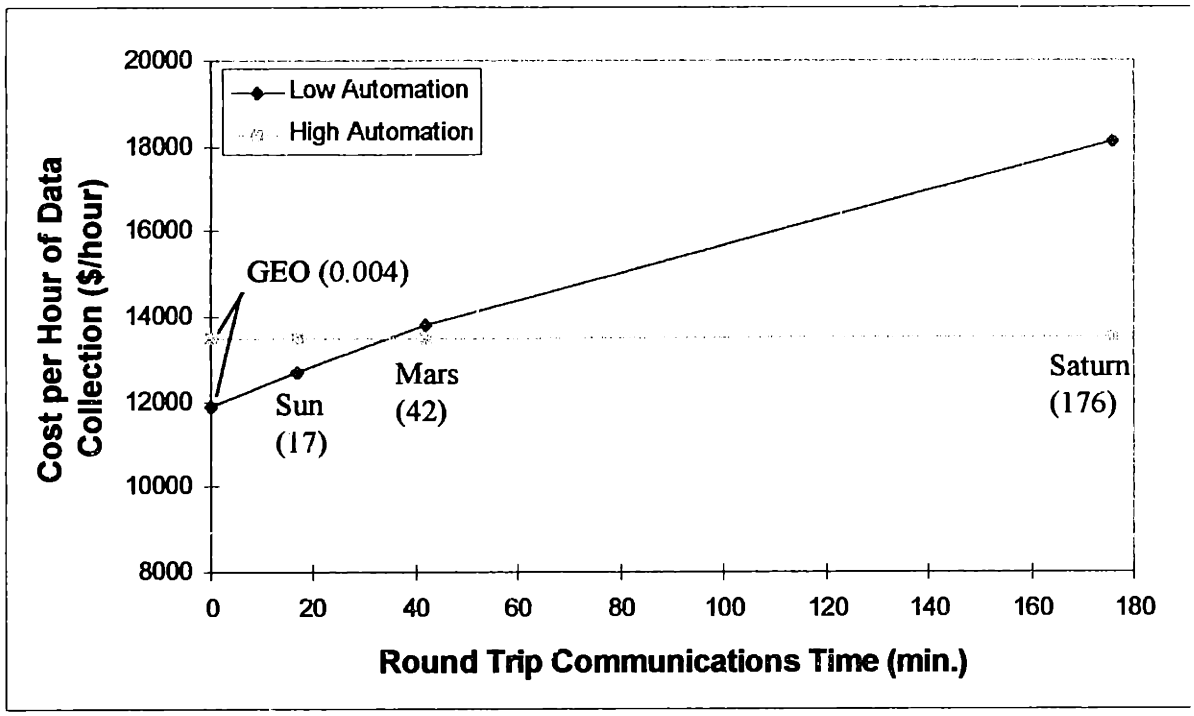


*Figure 6.23: Cost per Hour of Data Collection vs. Round Trip Communications Time for Low and High Levels of Automation*

# 7. Conclusions

## 7.1 Overview

A methodology has been presented that quantifies the effects of automation through an availability-based approach that captures both direct and opportunity costs in the calculation of life cycle costs. The end-to-end system is decomposed into functions, where each function's event and failure recovery processes are represented through Markov models. Each transition in the Markov model is linked to processor reliabilities and mean times to completion of tasks, as well as the reliability of the hardware components and the level of automation. The independent reliabilities of the functions are calculated, and the results are modified to account for dependencies among functions. From these availabilities, the overall system's mission objective availabilities are determined, thus producing the opportunity costs which are added to production, development, and operating costs.

A software tool (SOCRATES) has been written to allow system engineers to enter cost and reliability data and to specify a fault tree representation of the system architecture through a graphical user interface. The tool allows the user to interactively vary the levels of automation for each function, and resulting costs and reliabilities can be plotted for comparison. As more advanced human and software reliability and cost models are made available, they can be incorporated into SOCRATES since the tool has been written in a modular manner. This will allow the tool to mature during later stages of refinement.

## 7.2 Capabilities

As stated above, the methodology allows system engineers to estimate the effects that automating any function will have on the overall system availability and life cycle cost. This in turn will lead to more confidence in making decisions in an industry that is traditionally conservative as a result of the high risks and costs involved. The methodology is also general enough to allow the analyst to model the system at various levels. This allows the tool to be used to perform automation trade studies at various stages in the system design, and to varying levels of detail.

The case studies of Chapter 6 have demonstrated the use of SOCRATES to look at system-level trades associated with automating various systems. The model was used to investigate the automation of a simple function of a single satellite in order to minimize the life cycle cost metric. Automation was shown to be beneficial in this case. A second study investigated how the optimal level of automation for a complex function can vary with the number of satellites in a constellation. A critical constellation size was found above which automation resulted in the minimum life cycle costs. Below that point, automation did not pay off.

The final case study investigated the effects of increased delay times in satellite operations. A new metric was defined, the cost per hour of data collection, which was used for a scientific deep space mission. The levels of automation of the bus and payload were varied independently, and the delay time was varied from 0.25 minutes (GEO) to 176 minutes (Saturn). The study showed that it does not benefit to automate functions with mean times to failure much larger than the delay time. However, as the mean time to failure was reduced to be on the order of the delay time, automation reduced the cost per hour of data collection. Thus, for the bus function (large mean time to failure), no automation was optimal, while for the payload function (small mean time to failure), no automation was appropriate for near Earth missions, but more automation was optimal for more distant missions.

Future versions of the software tool will make extensive use of databases to allow baseline systems to be modified with little additional effort. In addition, the software tool has been written in a modular manner that allows for modification of the cost model as well as human and software reliability models. This will allow the basic backbone of the tool to be re-used and adapted as higher-fidelity models become available.

## 7.3 Limitations

Although some functional interdependence has been included which accounts for failures in one function creating failures/events in other functions, a deeper level of dependence is not yet modeled. Cases where one function's current state changes another's transition probabilities (but not to 1 or 0) cannot be handled by the model. Also, the dependencies need to be unidirectional. That is, if Function A depends on Function B, the converse cannot be true as currently modeled.

Another limitation to the SOCRATES implementation is that objective availability redundancy strings are required to be fully independent of each other. Each function of each string must be independent of the functions of any other redundancy string of that objective.

The methodology also assumes that permanent failures are independent of the current state of the system. This precludes cases where hardware fails as a result of a functional failure, as would be the case if an attitude control failure pointed a sensitive instrument into the sun, or if a thermal failure (prolonged) caused certain components to prematurely fail.

The methodology also ignores certain political and timing constraints related to adding automation. Although the technical feasibility of the automation can be captured through the development costs, there is no direct way to model the impact that increased automation may have on the system timeline. Also, subjective constraints such as a desire

to automate to demonstrate technical superiority are not captured through the life cycle cost metric. Therefore, the metric would need to be used in addition to these subjective considerations when ultimately making any decision regarding the level of automation that should be applied to the system's functions.

The current version of SOCRATES also has the limitation of being somewhat cumbersome. The large amount of data required to define the system at the required fidelity is time consuming, and many inputs may not be known with confidence. The linking of historical databases will no doubt improve this limitation, but some amount of uncertainty in inputs will be inevitable since highly autonomous systems are rare, and thus the database will be limited. The use of sensitivity analyses can also help overcome the difficulties of predicting model inputs for futuristic systems that are outside the range of historical experience.

The model presented in this thesis represents a framework to investigate automation of satellite systems. Much effort has been made to keep the model as general as possible to accommodate the broad variety of space missions and operations concepts. The methodology attempts to focus on operations fundamentals rather than on any specific type of system. This has resulted in a general model which can be widely used, but which requires significant effort in terms of input definition. This may be improved by performing parametric studies of current and past systems to identify key drivers of cost and availability. Such studies can simplify the methodology by focusing on these drivers rather than on the entire system.

# References


[1] Congor, Robert Microcosm Autonomous Navigation System (MANS). Presentation, August, 1995.

[2] Tandler, John. "Automating the Operations of the ORBCOMM Constellation." 10th Annual AIAA/Utah State University Conference on Small Satellites.

[3] Hornstein, Rhoda Shalter. "On-Board Autonomous Systems: Cost Remedy for Small Satellites or Sacred Cow?" 46th International Astronautical Congress, Oslo, Norway, Oct. 2-6, 1995.

[4] Congor, Robert "Microcosm Autonomous Navigation System (MANS)." Microcosm, Inc., Torrance, CA, 1995.

[5] Collins, John T, Simon Dawson, & James R. Wertz. "Autonomous Constellation Maintenance System." 10th Annual AIAA/USU Conference on Small Satellites, 1996.

[6] Sheridan, Thomas. Telerobotics, Automation, and Human Supervisory Control. MIT Press, Cambridge, MA, 1992.

[7] Weiner, Andrew J., David A. Thurman, & Christine M. Mitchel. "Applying case-based reasoning to aid fault management in supervisory control." Proceedings from 1993 IEEE International Conference for Systems, Man, and Cybernetics, Vancouver, B.C. 1995.

[8] Dhillon, B.S. & Chanan Singh. Engineering Reliability: New Techniques and Applications. John Wiley & Sons, NYC, New York, 1981.

[9] Babcock, Philip S. IV, "An Introduction to Reliability Modeling of Fault-Tolerant Systems." CSDL-R-1899, The Charles Stark Draper Laboratory, Cambridge, MA, 1986.

[10] Weinstein, William, & Philip S. Babcock IV. "Safety Analysis of Ares." CSDL-R-2013, The Charles Stark Draper Laboratory, Cambridge, MA, 1987.

[11] Thuesen, G.J. & W.J. Fabrycky. Engineering Economics, Eighth Edition. Prentice Hall Inc., Englewood Cliffs, NJ 1993.

[12] Schwarz, Robert. SOCRATES v. 2.0 User's Manual. MIT, May, 1997.

[13] Nguyen, P., N. Lozzi, W. Galang, Dr. S. Hu, A. Sjovold, P. Young, Dr. S. Book, & J. Schmitz. Space and Missile Systems Center Unmanned Space Vehicle Cost Model. Seventh Edition. Space and Missile Systems Center/FMC, Los Angeles Air Force Base, CA, 1984.

[14] Hecht, Herbert, & Myron Hecht. "Reliability Prediction for Spacecraft." RADC-TR-85-229 Final Technical Report. Rome Air Development Center, 1985.

[15] Bayt, Robert, Greg Giffen, Andjelka Kelic, Chris McLain, Piero Miotto, David Oh, Robert Schwarz, & Sanjay Vakil. Renaissance: Global Broadband Communications System. Space Systems Design Final Report, Department of Aeronautics and Astronautics, MIT, 1996.

[16] Sorensen, Trevor C., Dean C. Oswald, Rodney M. Shook, and James Van Gaasbeck. "Spacecraft Autonomous Operations Experiment Performed During the Clementine Lunar Mission." AIAA Journal of Spacecraft and Rockets. Vol. 32, No. 6 1995.

[17] Meeting Minutes, New Millenium Program Science Working Group Meeting. Pasadena, CA, March, 3-5, 1997.

[18] Kreysig, Erwin. Advanced Engineering Mathematics. Sixth Edition. John Wiley & Sons, NYC, New York, 1988.

# Appendix A: Markov Modeling

Markov modeling can provide a powerful tool when analyzing a stochastic process. It has advantages over other modeling techniques such as fault tree analysis and Monte Carlo simulations for large systems or very dependable systems. Fault trees are difficult to generate for complex systems, while Monte Carlo simulations require large run times for dependable systems. As with other techniques, Markov modeling starts with a definition of the system in terms of a state diagram. For the purposes of this appendix, a weather example will be used. Three states will be defined to simplify the problem. The states will represent clear weather, cloudy but dry weather, and precipitation. Figure A.1 shows a diagram of the weather model. The arrows between states represent transitions from one state to another. A transition probability is assigned to each arrow representing the probability of the transition taking place. Note that these probabilities are conditional probabilities since they assume the current state and represent the probabilities of transitioning from that state to the others (and the probability of not transitioning to another state). These transition probabilities are defined over a period of time. For this example, a day will be used for the time step. Therefore, p12 would be the probability of tomorrow being cloudy given that today was clear. There is an assumption with Markov modeling that the state probabilities are dependent only on the current state, and not on the past. Therefore, the probability of tomorrow being cloudy does not depend on yesterday's weather, but only on today's.

Example transition probabilities for the model of Figure A.1 are given in Table A.1. Notice that the sum of the probabilities leaving a state is equal to one (including the probability of leaving and arriving at the same state). These state probabilities may be used to derive the discrete state equations for the Markov model. The state equations can be written in matrix form as in Eq. A.1.

$$\begin{pmatrix} P(1) \\ P(2) \\ P(3) \end{pmatrix}_{t+1} = \begin{bmatrix} p11 & p21 & p31 \\ p12 & p22 & p32 \\ p13 & p23 & p33 \end{bmatrix} \begin{pmatrix} P(1) \\ P(2) \\ P(3) \end{pmatrix}_{t} \tag{A.1}$$

Once the state probability vector has been defined for the initial condition, Eq. A.1 can be used repetitively to calculate the state probability vector for future time steps. This is fine if the model is small, and the time step is not very small compared to the total time span to be studied. However, this is not always the case. The matrix squaring technique can be used to deal with this computational problem. Applying Eq. A.1 twice gives Eq. A.2 where S represents the state probability vector, and T represents the state transition matrix. Thus, by calculating $T^2$, the state probability vector at time t+2 can be found directly. Likewise, squaring $T^2$ gives $T^4$ which can be used to directly calculate S at time t+4. Squaring this leads to $S_{t+8}$, and so on. Thus, rather than performing a matrix multiplication, the matrices are squared using the last "squared" matrix for the next step. This can lead to a calculation of S for times $t+2^n$ where n represents the $n^{th}$ squaring of a matrix.

$$S_{t+2} = T \cdot S_{t+1} = T \cdot (T \cdot S_t) = T^2 \cdot S_t \tag{A.2}$$

Matrix squaring can greatly reduce the calculation times, but when implemented by a digital computer can also lead to numerical problems. This arises if the state probabilities are very small or very close to unity for many time steps. Roundoff error increases until the probability of leaving a state no longer exactly equals one. This error then blows up very quickly. This can be countered by adding a routine after each squaring which re-calculates the probability of staying in a state as 1 - the sum of probabilities of leaving that state. This helps to keep the roundoff error in bounds.

*Table A.1: Transition Probabilities for the Weather Model*

| | | |
|---|---|---|
| p11 = 0.6 | p21 = .2 | p31 = .5 |
| p12 = 0.3 | p22 = .3 | p32 = .1 |
| p13 = 0.1 | p23 = .5 | p33 = .4 |

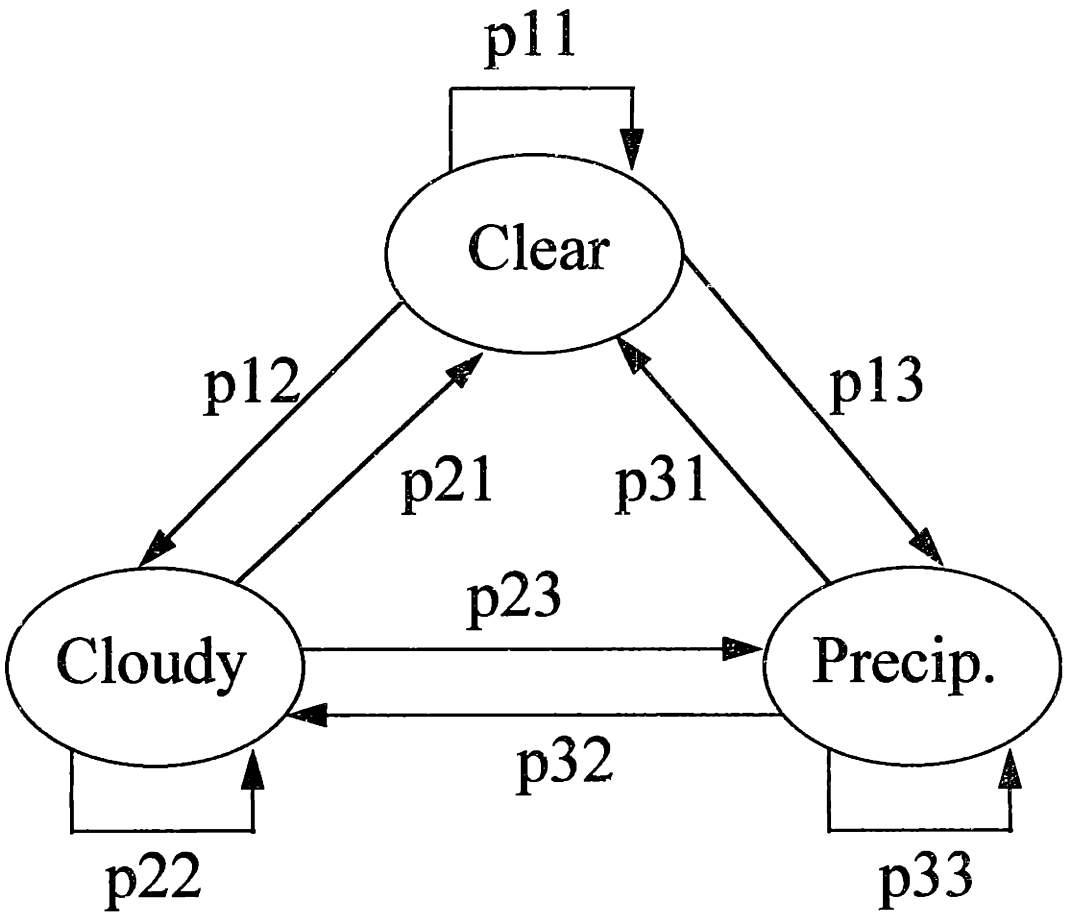


*Figure A.1: Markov Model of Weather*

# Appendix B: Calculation of Mean Time to Transition From a Series Combination of Events

The collapse of the discrete recovery process Markov model into the 4-State model is accomplished by representing a string or chain of transitions by a single transition. Each transition in general is described by a probability of transitioning at all (probability of correctly performing the task) and a probability of doing so within a given time step. Clearly, the probability of successfully transitioning from state 1 to state 4 is the product of the individual transition probabilities. If the probability of successfully transitioning from state **i** to state **j** is represented by $s_{i,j}$, then the success probability from state 1 to state 5 is given by Eq. B.1.

$$s_{1,5} = s_{1,2} \cdot s_{2,3} \cdot s_{3,4} \cdot s_{4,5} \tag{B.1}$$

The calculation of the overall transition rate is not as straightforward. Figure B.1 shows the discrete recovery processes in the top figure being reduced to a two state model below. The $\theta$'s represent the mean times to transition for each step. However, the mean time to transition from state 1 to state 5 is not exactly the sum of the mean times to failure for each transition. In fact, the probability distribution function for the total transition from state 1 to state 5 does not even follow an exponential distribution since the recovery processes must occur sequentially.

In order to take advantage of Markov modeling techniques, past independence must be preserved. This effectively limits the probability distribution of transitions to the exponential distribution. Thus, the goal is to find a mean time to transition $\theta$ which accurately predicts the transition from state 1 to state 5. The collapse to the 4 state model is essentially based upon collapsing such chains into single transitions. What follows is a

derivation of the actual distribution of the transition time and a comparison of this with the approximate exponential distribution.

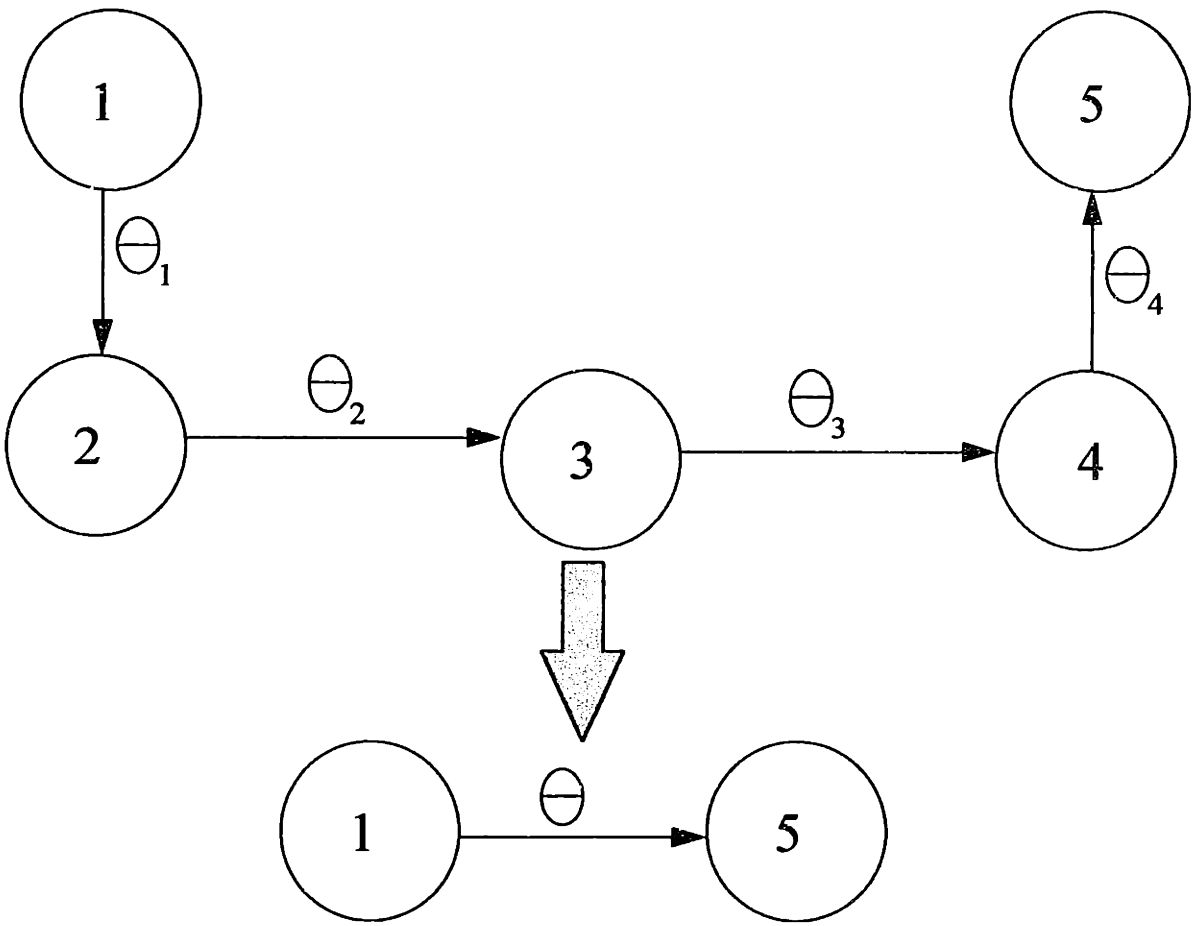


*Figure B.1: Collapse of a Markov Chain*

In words, the probability that a transition occurs in time t is the probability that all four transitions take place in time t, but sequentially. If the first transition takes t1 seconds, the next three transitions must take place in (t-t1) seconds. This is shown graphically in Figure B.2. The transition from state 1 to state 2 takes place in (t - t1) seconds leaving t1 seconds for the next three transitions. Likewise, the transition from state 2 to 3 takes place in (t1 - t2) seconds leaving t2 seconds for the remaining two transitions. In general, the probability of transitioning from state 1 to state 5 in time t is the sum of probabilities of the four transitions taking place such that the sum of their individual transition times is exactly t.

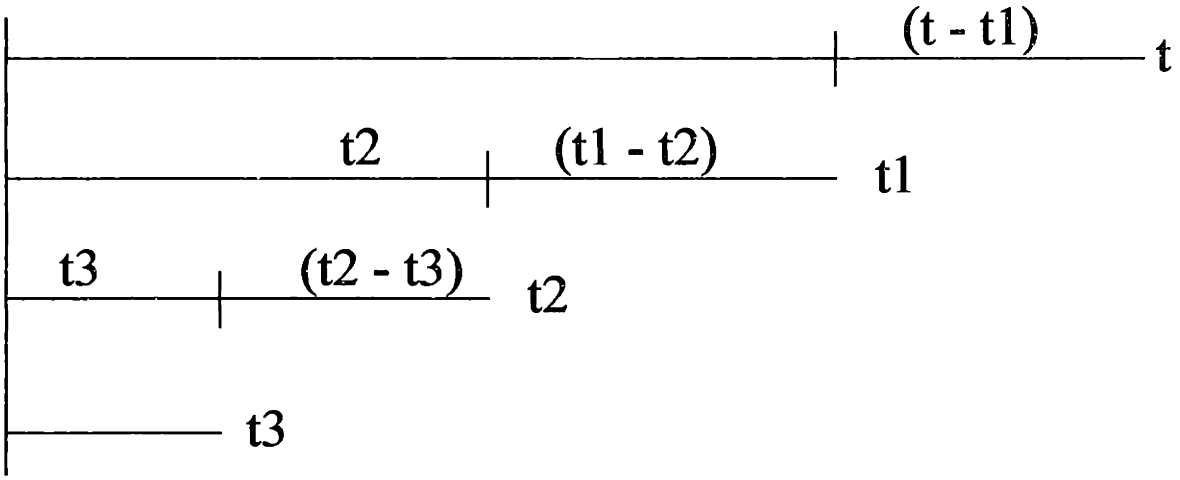


*Figure B.2: Transition Time Breakdown*

The graphical representation of Figure B.2 is characteristic of a convolution. The probability distribution function for the exponential distribution is given by Eq. B.2 where $\theta_n$ is the mean time to transition for the $n^{th}$ transition.

$$p(t) = \frac{1}{\theta_n} \cdot e^{-\left(\frac{t}{\theta_n}\right)} \tag{B.2}$$

The probability of transitioning within time t is then calculated by integrating Eq. B.2 with respect to time. Both the probability distribution function (pdf) and the transition probability (P(t)) are plotted in Figure B.3. The time axis has been normalized by $\theta_n$.

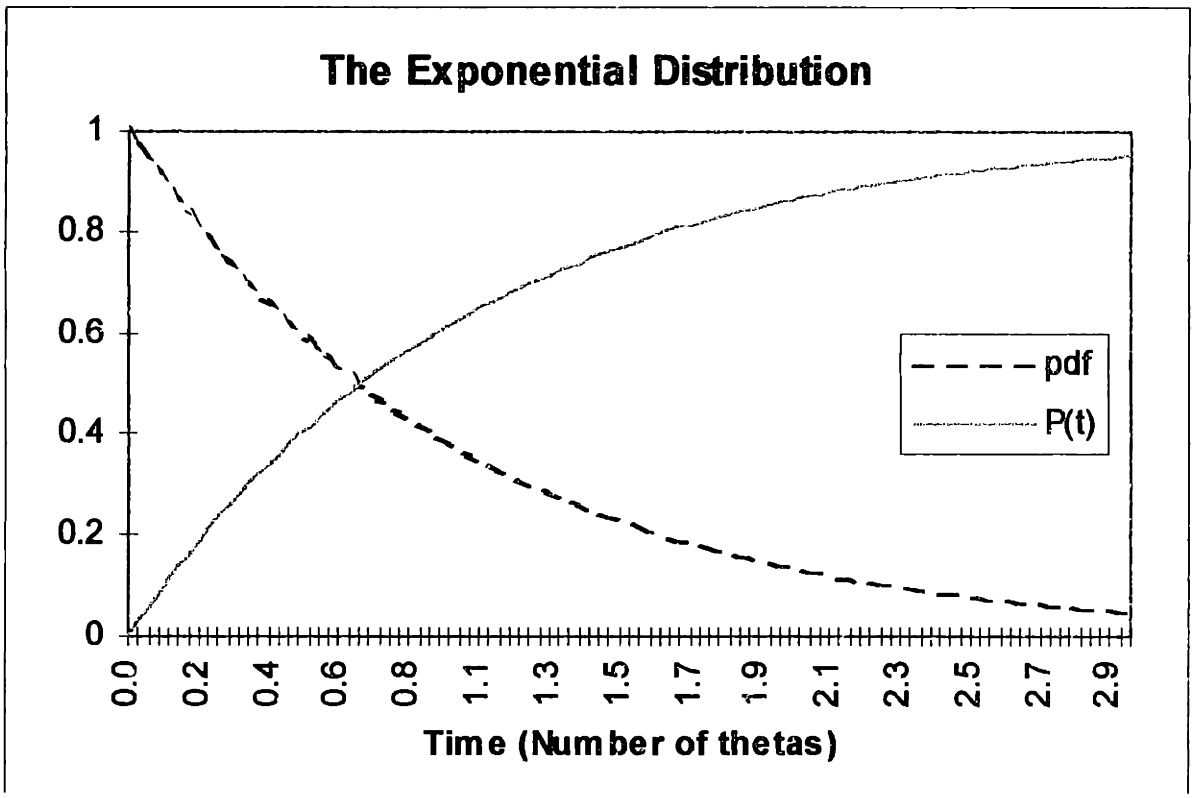


*Figure B.3: The Exponential Distribution*

Assuming the transitions all follow the exponential distribution of Figure B.3, the transition probability from state 1 to 5 is given by the multiple convolution integrals of Eq. B.3.

$$P(t) = \int_0^t \frac{1}{\theta_4} e^{-\frac{t-\tau_3}{\theta_4}} \int_0^{\tau_3} \frac{1}{\theta_3} e^{-\frac{\tau_3-\tau_2}{\theta_3}} \int_0^{\tau_2} \frac{1}{\theta_2} e^{-\frac{\tau_2-\tau_1}{\theta_2}} \frac{1}{\theta_1} e^{-\frac{\tau_1}{\theta_1}} d\tau_1 d\tau_2 d\tau_3 \qquad \text{(B.3)}$$

The result is given by

$$P(t) = c_1 e^{\left(\beta-\frac{1}{\theta_3}\right)t} + c_2 e^{\left(-\frac{1}{\theta_3}\right)t} + c_3 e^{\left(\gamma-\frac{1}{\theta_3}\right)t} + c_4 e^{-\frac{t}{\theta_4}} \qquad \text{(B.4)}$$

with the parameters defined by:

$$c_1 = \frac{\alpha}{\beta\rho\theta_4} \qquad c_2 = \frac{\alpha}{\eta\theta_4}\left(\frac{1}{\gamma} - \frac{1}{\beta}\right) \qquad c_3 = \frac{-\alpha}{\gamma\phi\theta_4}$$

$$c_4 = -(c_1 c_2 c_3) \qquad \alpha = \frac{1}{\theta_3}\left(\frac{1}{\theta_2 - \theta_1}\right)$$

$$\beta = \left(\frac{\theta_2 - \theta_3}{\theta_2\theta_3}\right) \qquad \gamma = \left(\frac{\theta_1 - \theta_3}{\theta_1\theta_3}\right)$$

$$\rho = \frac{1}{\theta_4} + \beta - \frac{1}{\theta_3} \qquad \phi = \frac{1}{\theta_4} + \gamma - \frac{1}{\theta_3} \qquad \eta = \frac{1}{\theta_4} - \frac{1}{\theta_3}$$

However, the resulting probability distribution function of Eq. B.4 does not lend itself readily for use in a Markov model. Markov models assume past independence. In effect, they imply a purely exponential distribution. The actual distribution more closely resembles the geometric distribution. The following plots compare the actual transition probability with an approximate exponential distribution (Eq. B.5) with a mean time to transition calculated as the sum of the individual transitions as indicated in Figure B.1.

$$\theta = \theta_1 + \theta_2 + \theta_3 + \theta_4 \tag{B.5}$$

The goodness of the approximation depends on the relative magnitudes of the transitions $\theta_1$ - $\theta_4$. Figure B.4 shows the probability distribution function for the actual case as well as the exponential approximation which results when the transitions are of similar size. The general behavior is captured by the approximation, but there is a significant discrepancy for small times. This is because the actual process requires a chain of four events to take place in order. The probability that this occurs instantly is zero. The exponential distribution allows for the entire process to occur very quickly. This error will

manifest itself in the transition probabilities of the Markov model. The exponential distribution results in the definition of a mean time to transition. Any deviation from the exponential distribution introduces a time variance of the transition rate. Figure B.5 plots the probability of transitioning in a time t.

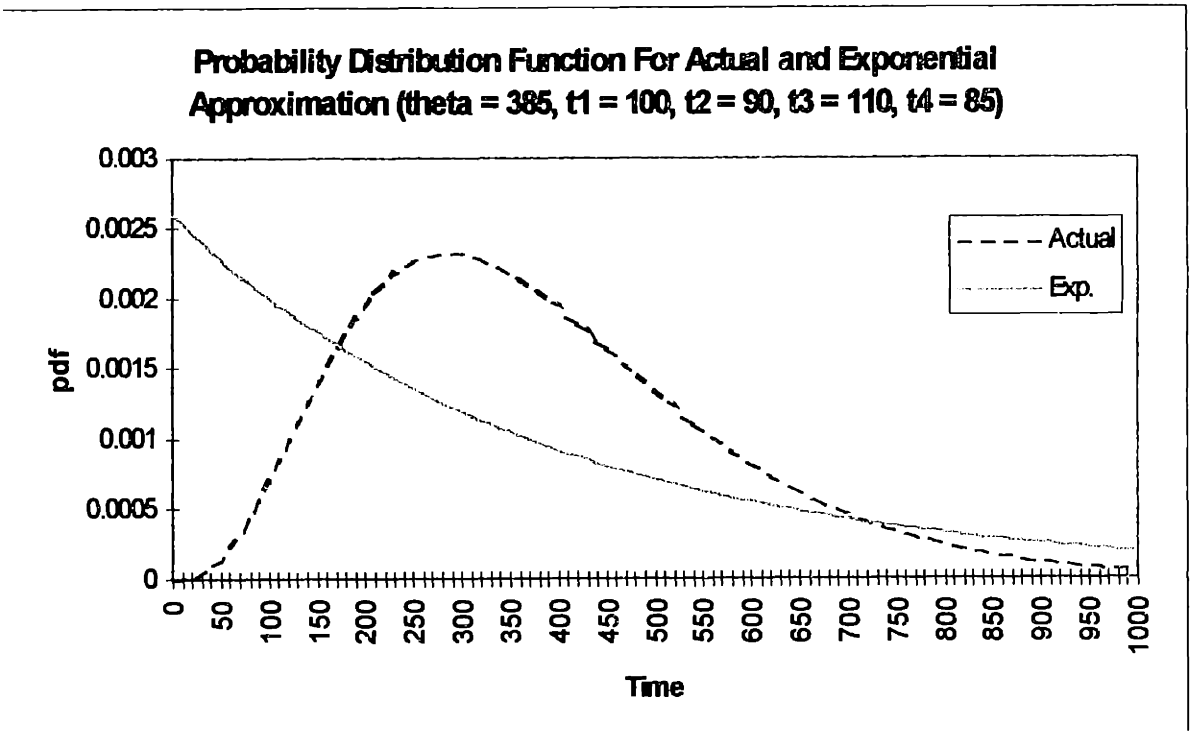


*Figure B.4: Probability Distribution Functions when Transition Times are Similar*

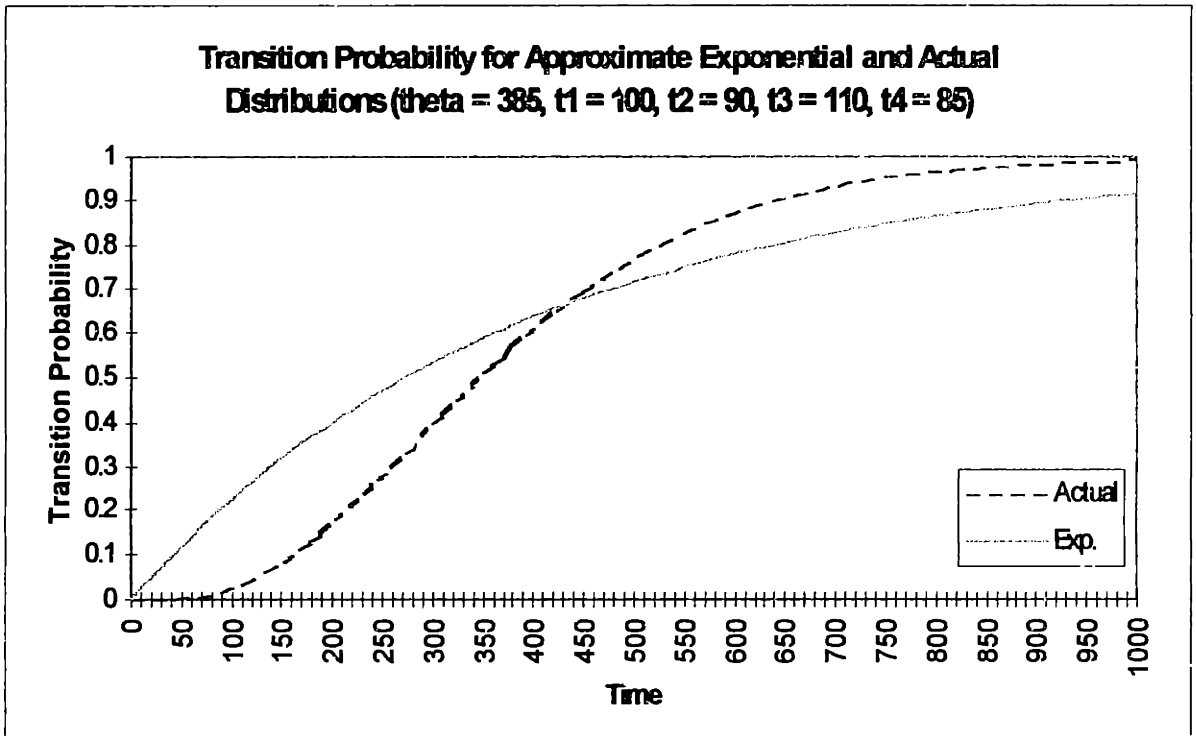


*Figure B.5: Transition Probabilities when Transition Times are Similar*

However, it is not obvious what the effect of assuming a time varying transition as constant will have on the system availability, which is the useful output of the functional availability model. Figure B.2 shows the calculated availability for both a discrete recovery process model and a single model with a mean time to recovery set to the sum of the individual recovery process transition times. The plot shows that the single transition model initially overestimates the availability due to the overestimate of the recovery transition probability. However, as time passes, and the system reaches steady state, the effects of the approximation decrease, and the curves approach each other with a very small error. Since the Markov models in this methodology are only used to determine the steady state value for the functional availabilities, the collapse of the recovery processes into a single transition actually introduces no error, even though some error occurs during the transitory period.

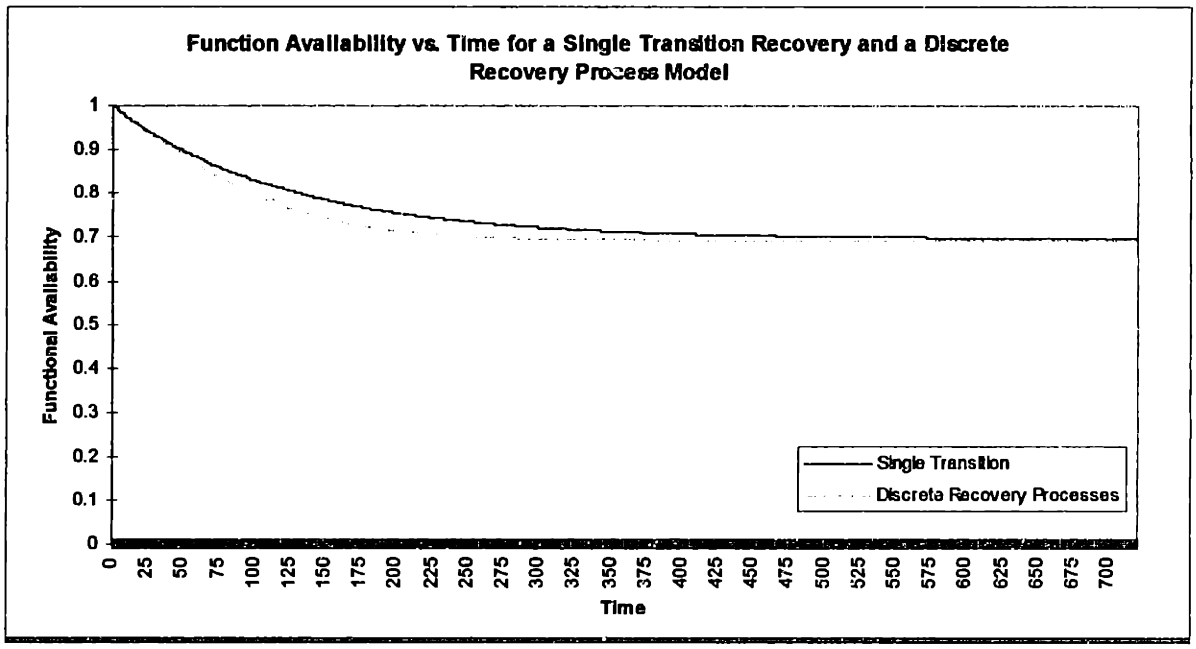


*Figure B.6: Comparison of the Function Availability for the Discrete Recovery Process Model and the Single Transition Model*

# Appendix C: Transition Probability Equations

The following sections include an algorithm for the equations, an equation for the calculation of the success probability, and an equation for the mean time to transition using the variable definitions in Table C.1.

*Table C.1: Variables for the Determination of Transition Probabilities*

| | | | |
|---|---|---|---|
| P | The transition state probability | mttx | The overall mean time to trans. |
| $P_S$ | Space processor success prob. | $mtt_S$ | Space processor transition time |
| $P_G$ | Ground processor success prob. | $mtt_G$ | Ground processor transition time |
| $P_H$ | Human operator success prob. | $mtt_H$ | Human operator transition time |
| $P_{AR}$ | Remote automation success prob. | $mtt_{AR}$ | Remote automation transition time |
| $P_{AG}$ | Ground automation success prob. | $mtt_{AG}$ | Ground automation transition time |

**LOA = (1, NA): Full Remote Automation**

Recovery requires only the space processor.

$$P = P_S$$

$$mttx = mtt_S \tag{C.1}$$

**LOA = (2, 1): Remote Paging, Full Ground Automation**

Recovery requires the space processor or both the ground processor and the remote automation.

$$P = 1 - \left[(1 - P_S)(1 - P_G \cdot P_{AR})\right]$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtf_S} + \frac{1 - P_S}{(mtt_G + mtt_{AR} + mtt_S)}\right]} \tag{C.2}$$

**LOA = (2, 2): Remote Paging, Ground Paging**

Recovery requires the space processor or (ground processor and remote automation) or (human operator and remote automation and ground automation)

$$P = 1 - \left[\left(1 - P_S\right)\left(1 - P_G \cdot P_{AR}\right)\left(1 - P_H \cdot P_{AR} \cdot P_{AG}\right)\right] \tag{C.3}$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{\left(1 - P_S\right) \cdot P_{AR} \cdot P_G}{\left(mtt_G + mtt_S + mtt_{AR}\right)} + \frac{\left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_G\right)}{\left(mtt_S + mtt_{AR} + mtt_G + mtt_{AG} + mtt_H\right)}\right]}$$

**LOA = (2, 3): Remote Paging, Ground Supervision**

Recovery requires the space processor or (ground processor and remote automation) or (human operator and remote automation and ground automation)

$$P = 1 - \left[\left(1 - P_S\right)\left(1 - P_G \cdot P_{AR}\right)\left(1 - P_H \cdot P_{AR} \cdot P_{AG}\right)\right] \tag{C.4}$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{\left(1 - P_S\right) \cdot P_{AR} \cdot P_G}{\left(mtt_G + mtt_S + mtt_{AR}\right)} + \frac{\left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_G\right)}{\left(mtt_S + mtt_{AR} + mtt_G + mtt_{AG} + mtt_H\right)}\right]}$$

**LOA = (2, 4): Remote Paging, Ground Cueing**

Recovery requires the space processor or the remote automation and the ground automation and either the human operator or the ground processor.

$$P = 1 - \left\{\left(1 - P_S\right) \cdot \left[1 - P_{AR} \cdot P_{AG} \cdot \left(1 - \left(1 - P_G\right) \cdot \left(1 - P_H\right)\right)\right]\right\} \tag{C.5}$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{\left(1 - P_S\right) \cdot P_{AR} \cdot P_G}{\left(mtt_G + mtt_{AR} + mtt_S\right)} + \frac{\left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_G\right)}{\left(mtt_S + mtt_{AR} + mtt_G + mtt_{AG} + mtt_H\right)}\right]}$$

**LOA = (2, 5): Remote Paging, Ground Data Filtering**

Requires the space processor or (the remote automation and the ground automation and the human operator).

$$P = 1 - \left[\left(1 - P_S\right) \cdot \left(1 - P_{AR} P_{AG} P_H\right)\right]$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{\left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_{AG} + mtt_H\right)}\right]} \tag{C.6}$$

**LOA = (2, 6): Remote Paging, No Ground Automation**

Requires the space processor or both the remote automation and the human operator.

$$P = 1 - \left[\left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_H\right)\right]$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{\left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_H\right)}\right]} \tag{C.7}$$

**LOA = (3, 1): Remote Supervision, Full Ground Automation**

Requires the space processor or both the remote automation and the ground processor.

$$P = 1 - \left[\left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_G\right)\right]$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{\left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_G\right)}\right]} \tag{C.8}$$

**LOA = (3, 2): Remote Supervision, Ground Paging**

Requires the space processor or both the remote automation and ground processor or both the remote automation, the ground automation, and the human operator.

$$P = 1 - \left[\left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - P_{AG} \cdot P_H\right)\right] \tag{C.9}$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{P_{AR} \cdot P_G \cdot \left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_G\right)} + \frac{\left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_G + mtt_{AG} + mtt_H\right)}\right]}$$

**LOA = (3, 3): Remote Supervision, Ground Supervision**

Requires the space processor or both the remote automation and ground processor or both the remote automation, the ground automation, and the human operator.

$$P = 1 - \left[\left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - P_{AG} \cdot P_H\right)\right] \tag{C.10}$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{P_{AR} \cdot P_G \cdot \left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_G\right)} + \frac{\left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_G + mtt_{AG} + mtt_H\right)}\right]}$$

**LOA = (3, 4): Remote Supervision, Ground Cueing**

Requires space processor or (remote automation and ground automation and either ground processor or human operator)

$$P = 1 - \left\{\left(1 - P_S\right) \cdot \left\{1 - P_{AR} \cdot P_{AG} \cdot \left[1 - \left(1 - P_G\right) \cdot \left(1 - P_H\right)\right]\right\}\right\}$$

$$mttx = \frac{1}{\left[\frac{P_S}{mtt_S} + \frac{\left(1 - P_S\right)}{\left(mtt_S + mtt_{AR} + mtt_G + mtt_{AG} + mtt_H\right)}\right]} \tag{C.11}$$

**LOA = (3, 5): Remote Supervision, Ground Data Filtering**

Requires space processor or (remote automation and ground automation and human operator).

$$P = 1 - \left[ (1 - P_S) \cdot (1 - P_{AR} \cdot P_{AG} \cdot P_H) \right]$$

$$mttx = \frac{1}{\left[ \frac{P_S}{mtt_S} + \frac{(1 - P_S)}{(mtt_S + mtt_{AR} + mtt_{AG} + mtt_H)} \right]} \qquad \text{(C. 12)}$$

**LOA = (3, 6): Remote Supervision, No Ground Automation**

Requires space processor or both the remote automation and human operator.

$$P = 1 - \left[ (1 - P_S) \cdot (1 - P_{AR} \cdot P_H) \right]$$

$$mttx = \frac{1}{\left[ \frac{P_S}{mtt_S} + \frac{(1 - P_S)}{(mtt_S + mtt_{AR} + mtt_H)} \right]} \qquad \text{(C.13)}$$

**LOA = (4, 1): Remote Cueing, Full Ground Automation**

Requires remote automation and either the space processor or the ground processor.

$$P = P_{AR} \cdot \left[ 1 - (1 - P_S) \cdot (1 - P_G) \right]$$

$$mttx = mtt_S + mtt_{AR} + mtt_G \qquad \text{(C.14)}$$

**LOA = (4, 2): Remote Cueing, Ground Paging**

Requires remote automation and (space processor or ground processor or both ground automation and human operator).

$$P = P_{AR}\left\{1-(1-P_S)\cdot\left\{1-\left[1-(1-P_G)\cdot(1-P_{AG}\cdot P_H)\right]\right\}\right\} \tag{C.15}$$

$$\text{mttx} = \frac{1}{\left[\dfrac{P_S}{(\text{mtt}_S+\text{mtt}_{AR})}+\dfrac{(1-P_S)}{(\text{mtt}_S+\text{mtt}_{AR}+\text{mtt}_G+\text{mtt}_{AG}+\text{mtt}_H)}\right]}$$

**LOA = (4, 3): Remote Cueing, Ground supervision**

Requires remote automation and (space processor or ground processor or both ground automation and human operator).

$$P = P_{AR}\left\{1-(1-P_S)\cdot\left\{1-\left[1-(1-P_G)\cdot(1-P_{AG}\cdot P_H)\right]\right\}\right\} \tag{C.16}$$

$$\text{mttx} = \frac{1}{\left[\dfrac{P_S}{(\text{mtt}_S+\text{mtt}_{AR})}+\dfrac{(1-P_S)}{(\text{mtt}_S+\text{mtt}_{AR}+\text{mtt}_G+\text{mtt}_{AG}+\text{mtt}_H)}\right]}$$

**LOA = (4, 4): Remote Cueing, Ground Cueing**

Requires remote automation and (space processor or (ground automation and either ground processor or human operator)).

$$P = P_{AR}\cdot\left\{1-(1-P_S)\cdot\left\{1-P_{AG}\cdot\left[1-(1-P_G)\cdot(1-P_H)\right]\right\}\right\} \tag{C.17}$$

$$\text{mttx} = \frac{1}{\left[\dfrac{P_S}{(\text{mtt}_S+\text{mtt}_{AR})}+\dfrac{(1-P_S)}{(\text{mtt}_S+\text{mtt}_{AR}+\text{mtt}_G+\text{mtt}_{AG}+\text{mtt}_H)}\right]}$$

**LOA = (4, 5): Remote Cueing, Ground Data Filtering**

Requires remote automation and either space processor or both ground automation and human operator.

$$P = P_{AR} \cdot \left[1 - \left(1 - P_S\right) \cdot \left(1 - P_{AG} \cdot P_H\right)\right]$$

$$mttx = mtt_S + mtt_{AR} + mtt_{AG} + mtt_H \tag{C.18}$$

**LOA = (4, 6): Remote Cueing, No Ground Automation**

Requires remote automation and either space processor or human operator.

$$P = P_{AR} \cdot \left[1 - \left(1 - P_S\right) \cdot \left(1 - P_H\right)\right]$$

$$mttx = mtt_S + mtt_{AR} + mtt_H \tag{C.19}$$

**LOA = (5, 1): Remote Data Filtering, Full Ground Automation**

Requires remote automation and ground processor.

$$P = P_{AR} \cdot P_G$$

$$mttx = mtt_{AR} + mtt_G \tag{C.20}$$

**LOA = (5, 2): Remote Data Filtering, Ground Paging**

Requires (remote automation and ground processor) or (remote automation and ground automation and human operator)

$$P = 1 - \left[\left(1 - P_{AR}P_G\right) \cdot \left(1 - P_{AR}P_{AG}P_H\right)\right]$$

$$mttx = \frac{1}{\left[\dfrac{P_G}{\left(mtt_{AR} + mtt_G\right)} + \dfrac{\left(1 - P_G\right)}{\left(mtt_{AR} + mtt_G + mtt_{AG} + mtt_H\right)}\right]} \tag{C.21}$$

**LOA = (5, 3): Remote Data Filtering, Ground Supervision**

Requires (remote automation and ground processor) or (remote automation and ground automation and human operator)

$$P = 1 - \left[\left(1 - P_{AR}P_{G}\right)\cdot\left(1 - P_{AR}P_{AG}P_{H}\right)\right]$$

$$mttx = \frac{1}{\left[\frac{P_{G}}{\left(mtt_{AR} + mtt_{G}\right)} + \frac{\left(1 - P_{G}\right)}{\left(mtt_{AR} + mtt_{G} + mtt_{AG} + mtt_{H}\right)}\right]} \qquad \text{(C.22)}$$

**LOA = (5, 4): Remote Data Filtering, Ground Cueing**

Requires remote automation, ground automation and either the ground processor or the human operator.

$$P = P_{AR}P_{AG}\left[1 - \left(1 - P_{G}\right)\cdot\left(1 - P_{H}\right)\right]$$

$$mttx = mtt_{AR} + mtt_{G} + mtt_{AG} + mtt_{H} \qquad \text{(C.23)}$$

**LOA = (5, 5): Remote Data Filtering, Ground Data Filtering**

Requires remote automation, ground automation, and the human operator.

$$P = P_{AR}P_{AG}P_{H}$$

$$mttx = mtt_{AR} + mtt_{AG} + mtt_{H} \qquad \text{(C.24)}$$

**LOA = (5, 6): Remote Data Filtering, No Ground Automation**

Requires remote automation and the human operator.

$$P = P_{AR}P_{H}$$

$$mttx = mtt_{AR} + mtt_{H} \qquad \text{(C.25)}$$

**LOA = (6, 1): No Remote Automation, Full Ground Automation**

Requires the ground prccessor.

$$P = P_G$$

$$mttx = mtt_G \tag{C.26}$$

**LOA = (6, 2): No Remote Automation, Ground Paging**

Requires the ground processor or both the ground automation and human operator.

$$P = 1 - (1 - P_G) \cdot (1 - P_{AG} P_H)$$

$$mttx = \frac{1}{\left[ \frac{P_G}{mtt_G} + \frac{(1 - P_G)}{(mtt_G + mtt_{AG} + mtt_H)} \right]} \tag{C.27}$$

**LOA = (6, 3): No Remote Automation, Ground Supervision**

Requires the ground processor or both the ground automation and human operator.

$$P = 1 - (1 - P_G) \cdot (1 - P_{AG} P_H)$$

$$mttx = \frac{1}{\left[ \frac{P_G}{mtt_G} + \frac{(1 - P_G)}{(mtt_G + mtt_{AG} + mtt_H)} \right]} \tag{C.28}$$

**LOA = (6, 4): No Remote Automation, Ground Cueing**

Requires ground automation and either the ground processor or human operator.

$$P = P_{AG} \cdot \left[ 1 - (1 - P_G) \cdot (1 - P_H) \right]$$

$$mttx = mtt_G + mtt_{AG} + mtt_H \tag{C.29}$$

**LOA = (6, 5): No Remote Automation, Ground Data Filtering**

Requires the ground automation and the human operator.

$$P = P_{AG}P_{H}$$

$$mttx = mtt_{AG} + mtt_{H} \tag{C.30}$$

**LOA = (6, 6): No Remote Automation, No Ground Automation**

Requires the human operator.

$$P = P_{H}$$

$$mttx = mtt_{H} \tag{C.31}$$

# Appendix D: Weighted Combination of Transition Times

This appendix provides the calculation of a mean time to completion of a task from a weighted combination of the mean times to completion of several modes by which the task may be completed. In practice, this is useful in determining the mean time to completion of a task which is nominally completed by a primary processor, but may alternatively (such as when the primary processor fails) be completed by a secondary processor.

The mean time to completion, mttc, can be converted into a transition probability during a time step, dt, by Eq. D.1.

$$P_{tx} = 1 - e^{-\left(\frac{dt}{mttc}\right)} \tag{D.1}$$

The exponential term may be expanded by Eq. D.2 [18].

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \cdots \tag{D.2}$$

Substitution of Eq. D.2 into Eq. D.1 yields Eq. D.3 which, for small dt reduces to Eq. D.4.

$$P = \frac{dt}{mttc} - \frac{1}{2}\left(\frac{dt}{mttc}\right)^2 + \frac{1}{6}\left(\frac{dt}{mttc}\right)^3 - \cdots \tag{D.3}$$

$$P \cong \frac{dt}{mttc} \tag{D.4}$$

Figure D.1 shows a diagram of the recovery modes once a failure has occurred. The left path has a success probability of P1 and a mean time to completion of mttc1. This corresponds to the path of the primary processor. In the event that the primary processor fails (1 - P1), the mean time to completion is the sum of the primary processor (it takes time to fail), and the mean time to completion of the secondary processor. The probability of transitioning from failed to non-failed is then the probability of successfully taking either path. The success probability is then given by Eq. D.5.

$$P = \frac{dt}{mttc} = 1 - \left[\left(1 - \frac{P1 \cdot dt}{mttc1}\right) \cdot \left(1 - \left(\frac{dt \cdot (1 - P1)}{(mttc1 + mttc2)}\right)\right)\right] \tag{D.5}$$

Expanding Eq. D.5 gives Eq. D.6.

$$P = \frac{dt}{mttc} = \frac{dt}{(mttc1 + mttc2)} + \frac{P1 \cdot dt}{mttc1} - \frac{P1 \cdot dt}{(mttc1 + mttc2)} + \frac{(P1^2 - P1) \cdot dt^2}{mttc1 \cdot (mttc1 + mttc2)} \tag{D.6}$$

Returning to the assumption that dt << mttc1 and dt << mttc2, the last term drops off. Combining terms with like denominators and solving for mttc gives

$$mttc = \frac{1}{\left[\frac{P1}{mttc1} + \frac{(1 - P1)}{(mttc1 + mttc2)}\right]} \tag{D.7}$$

The structure of Eq. D.7 is evident in the transition probability equations.

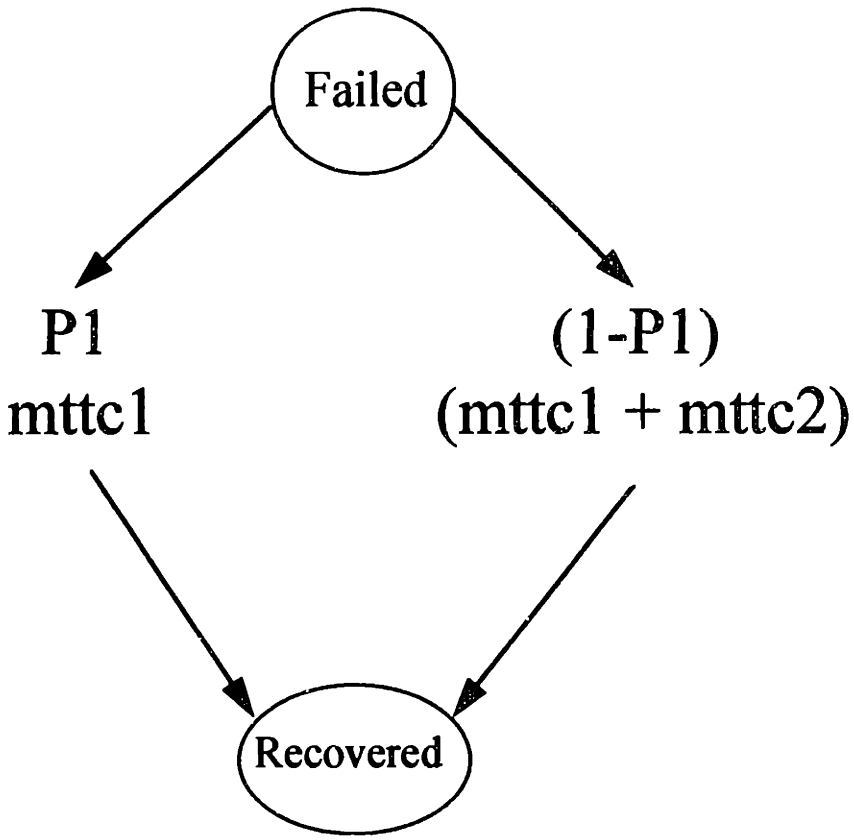


*Figure D.1: Completion Modes for a Failure*

# Appendix E: Calculation of the Overall Mean Time to Recovery from the State Probability Vector and the Overall Mean Time to Failure

It is assumed that a Markov process has been computed to calculate the steady state probability vector for a function. Since the probability of transitioning to the failed state is independent of the current non-failed state, the model may be collapsed to a two state model as shown in Figure E.1.

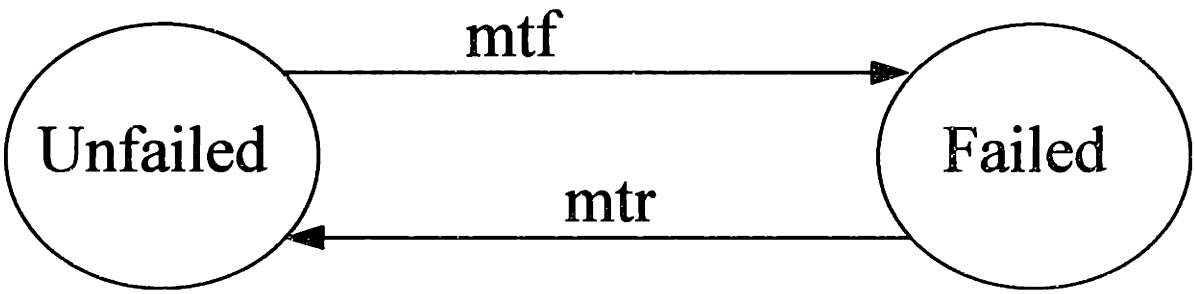


*Figure E.1: Collapsed Two State Markov Model*

From this model, the state equations may be written by Eq. E.1. It is assumed that the step time, dt, is small compared to the mean time to failure and mean time to failure, so the exponentials may be reduced to 1/transition time.

$$\begin{pmatrix} Op \\ F \end{pmatrix} = \begin{bmatrix} \left(1 - \frac{dt}{mtf}\right) & \frac{dt}{mtr} \\ \frac{dt}{mtf} & \left(1 - \frac{dt}{mtr}\right) \end{bmatrix} \begin{pmatrix} Op \\ F \end{pmatrix} \tag{E.1}$$

At steady state, the time variable vanishes from Eq. E.1, and the first equation may be used to define the steady state relationship among the spate probabilities and the transition rates.

$$Op = \left(1 - \frac{dt}{mtf}\right) \cdot Op + \frac{dt}{mtr} \cdot F \tag{E.2}$$

Rearranging Eq. E.2 to solve for the mean time to recovery yields Eq. E.3.

$$mtr = mtf \cdot \left(\frac{F}{Op}\right) \tag{E.3}$$

# Appendix F: Equations Converting Functional Failure Probabilities to Probabilities of the Human Performing the Recovery

## F.1 Equation Derivations

The information required to determine the probability that a human is involved in a recovery process at a given point in time has already been calculated in the functional availability model. However, no state had been defined as such, and therefore, this information must be extracted from the steady-state availability results. This is done by creating a Markov model including the new states, the knowledge of the transition probabilities, and the steady state probability vector. The 2 state model of Figure 3.5 forms the baseline, with the "failure" state being expanded to isolate the probability that each processor is working on the recovery process. The equations for the probability of a human performing the recovery differ depending on the level of automation, but the form of the equation only reflects two cases. One case where all three processors (space, ground, and human operator) are in the loop, and one case where only one computer processor (space or ground) and the human operator are in the loop. The case where only the human in the loop is a trivial case, and the probability of the human being in a recovery state is equal to that of the function being in the recovery state. The first two cases are a little more complicated. Note that what follows is valid for event recovery as well as failure recovery.

### *F.1.1 Two Processor Case*

For the case where there are two processors, a computer and a human, the failure state is split into two states, one which represents the computer processor attempting the recovery, and one which represents the human attempting the recovery. The resulting Markov model is shown in Figure F.1, and Table F.1 lists the variables used in the model.

*Table F.1: Two Processor Markov Model Variable Definitions*

| | |
|---|---|
| txf | Probability of transition "F" = failure. |
| tx1 | Probability of transition "1" = computer processor recovery |
| tx2 | Probability of transition "2" = human recovery |
| tx3 | Probability of transition "3" = computer failure to recover |
| Op | State probability of function being available |
| F | State probability of function being in failure recovery |
| CR | State probability of function being in a computer recovery state |
| HR | State probability of function being in a human recovery state |

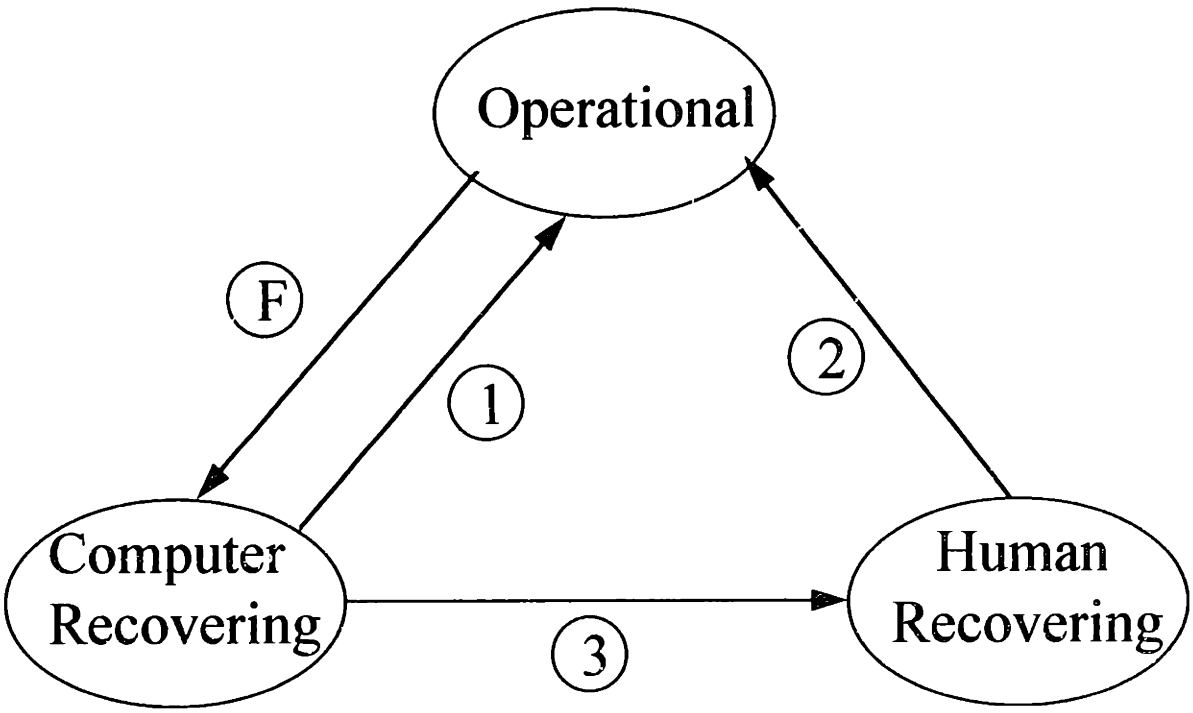


*Figure F.1: Two Processor Markov Model*

The Markov model of Figure F.1 is represented by the matrix equation of Eq. F.1. However, at steady state, the state vector at time t+1 is identical to the vector at time t, so the subscripts can be dropped.

$$\begin{pmatrix} Op \\ CR \\ HR \end{pmatrix}_{t+1} = \begin{bmatrix} (1-txf) & tx1 & tx2 \\ txf & (1-tx1-tx3) & 0 \\ 0 & tx3 & (1-tx2) \end{bmatrix} \begin{pmatrix} Op \\ CR \\ HR \end{pmatrix}_{t} \qquad (F.1)$$

From the functional availability model, the steady state availability (Op) is known. Also, the probability of being in a failure state is known (F = 1 - Op). The transition probabilities can be calculated from the processor inputs to the model, and the level of automation. Such relationships are given in section 0. What needs to be determined is the probability that the function is in a human recovery state since this is the state which results in human operations. The state equation for the Op state, along with the relationship: F = CR + HR, yields Eq. F.2.

$$Op = Op \cdot (1 - txf) + (F - HR) \cdot tx1 + HR \cdot tx2 \tag{F.2}$$

Rearranging Eq. F.2 to solve for HR yields Eq. F.3. It should be noted that txf is the failure transition probability, which is given by Eq. F.4 for a time step of dt. The state probabilities Op and F are obtained directly from the functional availability results, and the transitions tx1 and tx2 are given in section 0 as functions of the level of automation.

$$HR = \frac{Op \cdot txf - F \cdot tx1}{tx2 - tx1} \tag{F.3}$$

$$txf = 1 - e^{-\left(\frac{dt}{mtf}\right)} \tag{F.4}$$

### *F.1.2 Three Processor Case*

For the case where both computer processors and the human operator are in the loop, the failure state is divided into three separate states as shown in Figure F.2. The definitions for the model are given in Table F.2.

*Table F.2: Variable Definitions for the Three Processor Markov Model*

| | |
|---|---|
| txf | Probability of transition "F" = failure. |
| tx4 | Probability of transition "4" = space computer processor recovery |
| tx5 | Probability of transition "5" = space computer failure to recover |
| tx6 | Probability of transition "6" = ground computer processor recovery |
| tx7 | Probability of transition "7" = ground computer failure to recovery |
| tx8 | Probability of transition "8" = human recovery |
| Op | State probability of function being available |
| F | State probability of function being in failure recovery |
| S | State probability of function being in a space computer recovery state |
| G | State probability of function being in a ground computer recovery state |
| H | State probability of function being in a human recovery state |

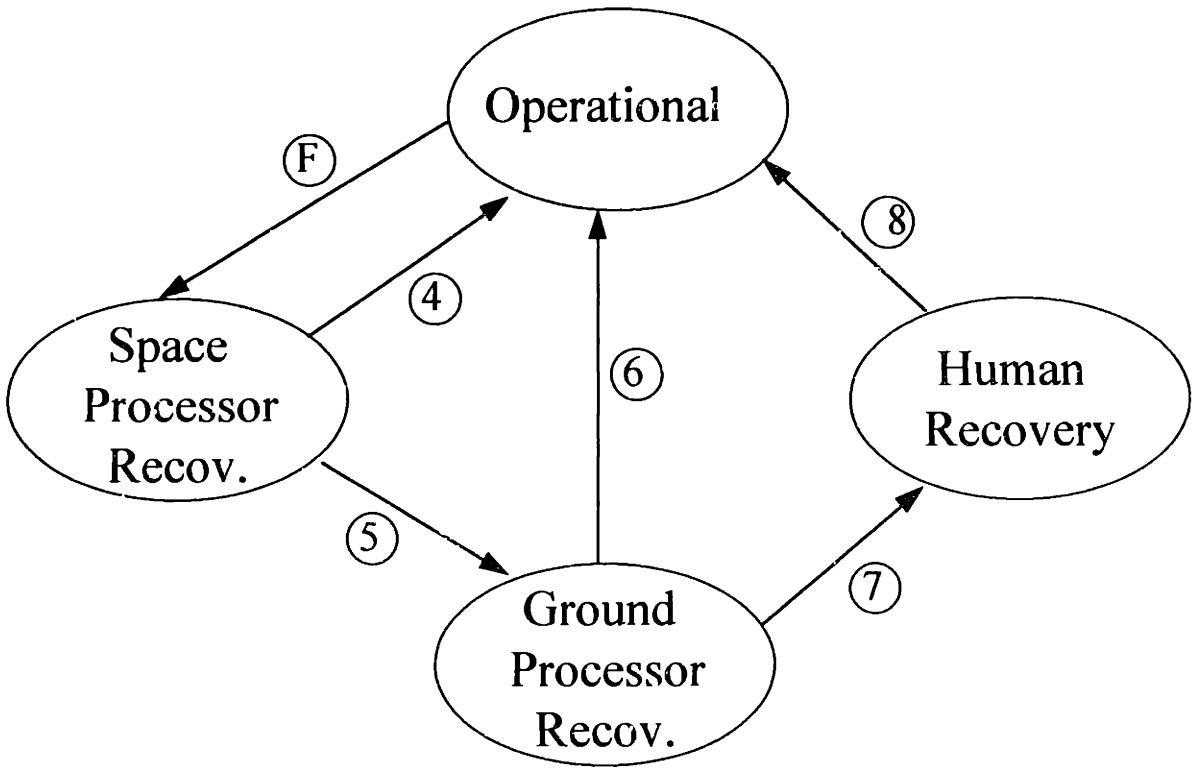


*Figure F.2: Three Processor Markov Model*

Again, the steady state matrix equations are generated directly from the model.

$$\begin{pmatrix} Op \\ S \\ G \\ H \end{pmatrix} = \begin{bmatrix} (1-txf) & tx4 & tx6 & tx8 \\ txf & (1-tx4-tx5) & 0 & 0 \\ 0 & tx5 & (1-tx6-tx7) & 0 \\ 0 & 0 & & (1-tx8) \end{bmatrix} \begin{pmatrix} Op \\ S \\ G \\ H \end{pmatrix} \tag{F.5}$$

There are now three unknowns: S, G, and H. The relationship F=S+G+H provides one equation. The operational state equation from Eq. F.5 provides a second. The third may come from any of the remaining three state equations. Here, the last equation is used since it has the simplest terms. Rearranging the last state equation to relate G to H yields

$$G = H \cdot \left(\frac{tx8}{tx7}\right) \tag{F.6}$$

Substituting Eq. F.6 into the first equation of Eq. F.5, and using the relationship among F, S, G, and H, and then rearranging to solve for H yields Eq. F.7.

$$H = \frac{Op \cdot txf - F \cdot tx4}{\left[\frac{(tx6-tx4)}{tx7} - tx4 + tx8\right]} \tag{F.7}$$

## F.2 Equations

The equations are dependent on the level of automation, and are as follows. As with the transition equations, the formulae for ground functions may be found by setting the level of remote automation to 6 (No Remote Automation). Rather than providing the formulae from the basic variables in Table C.1, the variables defined in section 0 are used, and their definitions in terms of the variables of Table C.1 are given along with the reference to which equation of section 0 which is to be used. Table F.3 restates those variables from Table C.1 which are used along with new variable definitions.

*Table F.3: Variable Definitions*

| | | | |
|---|---|---|---|
| Op | The function's availability | F | The function's probability of being failed |
| S | The space processor prob. of being in a recovery state. | G | The ground processor probability of being in a recovery state. |
| H | The human operator probability of being in a recovery state. | dt | Time step |
| $P_S$ | Space processor success prob. | $mtt_S$ | Space processor transition time |
| $P_G$ | Ground processor success prob. | $mtt_G$ | Ground processor transition time |
| $P_H$ | Human operator success prob. | $mtt_H$ | Human operator transition time |
| $P_{RA}$ | Remote automation success prob. | $mtt_{RA}$ | Remote automation transition time |
| $P_{GA}$ | Ground automation success prob. | $mtt_{GA}$ | Ground automation transition time |

**LOA = (1, NA): Full Remote Automation**

H=0, no human in the loop.

**LOA = (2, 1): Remote Paging and Full Ground Automation**

H=0, no human in the loop.

**LOA = (2, 2): Remote Paging and Ground Paging**

Use Eq. F.7 with the following definitions.

$$tx4 = P_S \cdot \left(1 - e^{-(dt/mtrs)}\right) \qquad tx5 = \left(1 - P_S\right) \cdot \left(1 - e^{-(dt/mtrs)}\right)$$

$$tx6 = P_{AR} \cdot P_G \cdot \left(1 - e^{-\left(\frac{dt}{(mtr_{AR} + mtr_G)}\right)}\right) \qquad tx7 = \left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_G}\right)}\right)$$

$$tx8 = P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (2, 3): Remote Paging and Ground Supervision**

Use Eq. F.7 with the following definitions.

$$tx4 = P_S \cdot \left(1 - e^{-(dt/mtrs)}\right) \qquad tx5 = \left(1 - P_S\right) \cdot \left(1 - e^{-(dt/mtrs)}\right)$$

$$tx6 = P_{AR} \cdot P_G \cdot \left(1 - e^{-\left(\frac{dt}{(mtr_{AR} + mtr_G)}\right)}\right) \qquad tx7 = \left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_G}\right)}\right)$$

$$tx8 = P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (2, 4): Remote Paging and Ground Cueing**

Use Eq. F.3 with the following definitions.

$$tx1 = P_S \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right) \qquad tx3 = (1 - P_S) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right)$$

$$tx2 = P_{AR} \cdot \left[1 - (1 - P_G) \cdot (1 - P_{AR})\right] \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G + mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (2, 5): Remote Paging and Ground Data Filtering**

Use Eq. F.3 with the following definitions.

$$tx1 = P_S \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right) \qquad tx3 = (1 - P_S) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right)$$

$$tx2 = P_{AR} \cdot P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (2, 6): Remote Paging and No Ground Automation**

Use Eq. F.3 with the following definitions.

$$tx1 = P_S \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right) \qquad tx3 = (1 - P_S) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right)$$

$$tx2 = P_{AR} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_H}\right)}\right)$$

**LOA = (3, 1): Remote Supervision and Full Ground Automation**

H=0, no human in the loop.

**LOA = (3, 2): Remote Supervision and Ground Paging**

Use Eq. F.7 with the following definitions.

$$tx4 = P_S \cdot \left(1 - e^{-(dt/mtrs)}\right) \qquad tx5 = \left(1 - P_S\right) \cdot \left(1 - e^{-(dt/mtrs)}\right)$$

$$tx6 = P_{AR} \cdot P_G \cdot \left(1 - e^{-\left(\frac{dt}{(mtr_{AR} + mtr_G)}\right)}\right) \qquad tx7 = \left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_G}\right)}\right)$$

$$tx8 = P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (3, 3): Remote Supervision and Ground Supervision**

Use Eq. F.7 with the following definitions.

$$tx4 = P_S \cdot \left(1 - e^{-(dt/mtrs)}\right) \qquad tx5 = \left(1 - P_S\right) \cdot \left(1 - e^{-(dt/mtrs)}\right)$$

$$tx6 = P_{AR} \cdot P_G \cdot \left(1 - e^{-\left(\frac{dt}{(mtr_{AR} + mtr_G)}\right)}\right) \qquad tx7 = \left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_G}\right)}\right)$$

$$tx8 = P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (3, 4): Remote Supervision and Ground Cueing**

Use Eq. F.3 with the following definitions.

$$tx1 = P_S \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right) \qquad tx3 = \left(1 - P_S\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right)$$

$$tx2 = P_{AR} \cdot \left[1 - \left(1 - P_G\right) \cdot \left(1 - P_{AR}\right)\right] \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G + mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (3, 5): Remote Supervision and Ground Data Filtering**

Use Eq. F.3 with the following definitions.

$$tx1 = P_S \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right) \qquad tx3 = \left(1 - P_S\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right)$$

$$tx2 = P_{AR} \cdot P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (3, 6): Remote Supervision and No Ground Automation**

Use Eq. F.3 with the following definitions.

$$tx1 = P_S \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right) \qquad tx3 = \left(1 - P_S\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S}\right)}\right)$$

$$tx2 = P_{AR} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_H}\right)}\right)$$

**LOA = (4, 1): Remote Cueing and Full Ground Automation**

H=0, no human in the loop.

**LOA = (4, 2): Remote Cueing and Ground Paging**

Use Eq. F.3 with the following definitions.

$$tx1 = \left[1 - \left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P \cdot G\right)\right] \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S + mtr_{AR} + mtr_O}\right)}\right)$$

$$tx2 = P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_H + mtr_{AO}}\right)}\right)$$

$$tx3 = \left(1 - P_S\right) \cdot \left(1 - P_{AR} \cdot P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_S + mtr_{AR} + mtr_O}\right)}\right)$$

**LOA = (4, 3): Remote Cueing and Ground Supervision**

Use Eq. F.3 with the following definitions.

$$tx1 = \left[1-\left(1-P_S\right)\cdot\left(1-P_{AR}\cdot P\cdot G\right)\right]\cdot\left(1-e^{-\left(\frac{dt}{mtr_S+mtr_{AR}+mtr_G}\right)}\right)$$

$$tx2 = P_{AG}\cdot P_H\cdot\left(1-e^{-\left(\frac{dt}{mtr_H+mtr_{AG}}\right)}\right)$$

$$tx3 = \left(1-P_S\right)\cdot\left(1-P_{AR}\cdot P_G\right)\cdot\left(1-e^{-\left(\frac{dt}{mtr_S+mtr_{AR}+mtr_G}\right)}\right)$$

**LOA = (4, 4): Remote Cueing and Ground Cueing**

H = F, human always in loop.

**LOA = (4, 5): Remote Cueing and Ground Data Filtering**

H = F, human always in loop.

**LOA = (4, 6): Remote Cueing and No Ground Automation**

H = F, human always in loop.

**LOA = (5, 1): Remote Data Filtering and Full Ground Automation**

H=0, no human in the loop.

**LOA = (5, 2): Remote Data Filtering and Ground Paging**

Use Eq. F.3 with the following definitions.

$$tx1 = P_G \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right) \qquad tx3 = \left(1 - P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right)$$

$$tx2 = P_{AR} \cdot P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (5, 3): Remote Data Filtering and Ground Supervision**

Use Eq. F.3 with the following definitions.

$$tx1 = P_G \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right) \qquad tx3 = \left(1 - P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right)$$

$$tx2 = P_{AR} \cdot P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AR} + mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (5, 4): Remote Data Filtering and Ground Cueing**

H = F, human always in loop.

**LOA = (5, 5): Remote Data Filtering and Ground Data Filtering**

H = F, human always in loop.

**LOA = (5, 6): Remote Data Filtering and No Ground Automation**

H = F, human always in loop.

**LOA = (6, 1): No Remote Automation and Full Ground Automation**

H=0, no human in the loop.

**LOA = (6, 2): No Remote Automation and Ground Paging**

Use Eq. F.3 with the following definitions.

$$tx1 = P_G \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right) \qquad tx3 = \left(1 - P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right)$$

$$tx2 = P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (6, 3): No Remote Automation and Ground Supervision**

Use Eq. F.3 with the following definitions.

$$tx1 = P_G \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right) \qquad tx3 = \left(1 - P_G\right) \cdot \left(1 - e^{-\left(\frac{dt}{mtr_G}\right)}\right)$$

$$tx2 = P_{AG} \cdot P_H \cdot \left(1 - e^{-\left(\frac{dt}{mtr_{AG} + mtr_H}\right)}\right)$$

**LOA = (6, 4): No Remote Automation and Ground Cueing**

H = F, human always in loop.

**LOA = (6, 5): No Remote Automation and Ground Data Filtering**

H = F, human always in loop.

**LOA = (6, 6): No Remote Automation and No Ground Automation**

H = F, human always in loop.

# Appendix G: Case Study Data Tables

## G.1 Tracking Automation of Single Satellite

The mission life for both the single geostationary satellite and the constellation was assumed to be 10 years with a 10% discount rate.

*Table G.1: Tracking Function Development Costs*

| Level of Automation | Equipment Required | Flight Software (lines of code) | Ground Software (lines of code) | Tracking Function Costs |
|---|---|---|---|---|
| No Remote or Ground Aut. | Radar (100% usage) | 0 | 2750 | $1,371,750 |
| No Rem. Aut., Ground Data Filtering | Radar (100% usage) | 0 | 2500 | $1,292,500 |
| Remote Cueing, Ground Data Filtering | Radar (100% usage), On-board navigation equip. | 2750 | 250 | $3,250,750 |
| Remote Paging, Ground Data Filtering | Radar (10% usage), On-board nav. equip. | 2750 | 250 | $2,800,750 |
| Remote Paging, Ground Cueing | Radar (10% usage), On-board nav. equip. | 2750 | 250 | $2,800,750 |
| Remote Paging, Ground Paging | Radar (10% usage), On-board nav. equip. | 2750 | 250 | $2,800,750 |
| Fully Automated | On-board nav. eq. | 2750 | 0 | $2,721,500 |

* Flight Software = $626 per line; Ground Software = $317 per line

*Table G.2: Operations Costs*

| | |
|---|---|
| Operator Base Salary | $50,000 per year |
| Operator Overhead (Admin., etc.) | 2.4 x Base Salary per operator per year |
| Maintenance Costs | $30,000 per ground function per year |
| Training Costs | $10,000 per operator per year |

*Table G.3: Single Satellite Interdependency Matrix*

| Independent Function / Dependent Function | AR | TR | CC | AC | PG | PD | TM | CH | OC | PRx | PTx | TC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Archiving | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Tracking | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Control Center Comm. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Attitude Control | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Power Generation | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Power Distribution | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Telemetry | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Command Handling | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Orbit Control | 0 | 4 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| Payload Receive | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| Payload Transmit | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| Thermal Control | 0 | 0 | 0 | 3 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

* Dependency Types:

Type 0: No dependency

Type 2: Failure of independent function causes unavailability of dependent function

Type 1: Event of independent function causes event of dependent function

Type 3: Failure of independent function causes event of dependent function

Type 4: Failure of independent function causes failure of dependent function

*Table G.4: Objective Requirements for Single Satellite*

| Objective | Requirements |
|---|---|
| Archiving (CC obj.) | Archiving, CC. Comm., Command Handling, Power Distribution, Telemetry |
| Tracking (CC obj.) | Tracking |
| Downlink (Sat. obj) | CC Comm., Payload Transmit, Orbit Control, Command Handling, Power Distribution, Power Generation, Attitude Control, Thermal Control |
| Uplink (Sat. obj) | CC Comm, Payload Receive, Orbit Control, Command Handling, power Distribution, Power Generation, Attitude Control, Thermal Control |
| Communications Link (M.O.)* | Uplink, Downlink |

*Communications Link Revenue = $38 Million per month

*Table G.5: Software and Human Task Reliabilities*

| Task Complexity | Description | Software Reliability | Human Reliability |
|---|---|---|---|
| Simple | Flown before | 0.9999 | 0.98 |
| Moderately Simple | Easy, not flown before | 0.999 | 0.999 |
| Moderately Complex | Hard but predictable | 0.99 | 0.999 |
| Complex | Hard, unpredictable | 0.9 | 0.99 |

*Table G.6: Automation Task Software Parameters and Human Supervisory Loads*

| Level of Automation | Code Reliability | Execution Time (min) | Human Supervisory Load (persons) |
|---|---|---|---|
| Fully Automated | N/A | N/A | 0 |
| Paging | 0.9999 | .1 | 0 |
| Supervision | 0.99999 | .1 | 0.33 |
| Cueing | 0.9999 | .2 | 0.5 |
| Data Filtering | 0.9999 | .2 | 0.5 |
| No Automation | N/A | N/A | 0.5 |

*Table G.7: Task Complexity and Completion Times for Event Recovery*

| Function | Complexity | Space Processor Completion Time (min.) | Ground Processor Completion Time (min.) | Human Operator Completion Time (min.) |
|---|---|---|---|---|
| Archiving | Mod. Simple | N/A | 1 | 30 |
| Tracking | Simple | N/A | N/A | 5 |
| Control Center Comm. | Mod. Simple | N/A | N/A | 2 |
| Attitude Control | Mod. Simple | 2 | N/A | 60 |
| Command Handling | Simple | N/A | 1 | 10 |
| Telemetry | Simple | 1 | N/A | N/A |
| Power Distribution | Mod. Simple | 1 | N/A | 20 |
| Power Generation | Mod. Simple | 1 | N/A | 30 |
| Orbit Control | Complex | N/A | N/A | 60 |
| Payload Receive | Complex | N/A | N/A | 60 |
| Payload Transmit | Complex | N/A | N/A | 60 |
| Thermal Control | Simple | 3 | 2 | 10 |

*Table G.8: Task Complexity and Completion Times for Failure Recovery*

| Function | Complexity | Space Processor Completion Time (min.) | Ground Processor Completion Time (min.) | Human Operator Completion Time (min.) |
|---|---|---|---|---|
| Archiving | Mod. Simple | N/A | 2 | 60 |
| Tracking | Simple | N/A | N/A | 10 |
| Control Center Comm. | Mod. Simple | N/A | N/A | 10 |
| Attitude Control | Mod Complex | N/A | N/A | 60 |
| Command Handling | Mod. Simple | N/A | N/A | 30 |
| Telemetry | Mod. Simple | N/A | N/A | 20 |
| Power Distribution | Mod. Simple | N/A | N/A | 30 |
| Power Generation | Simple | 1 | N/A | 20 |
| Orbit Control | Complex | N/A | N/A | 120 |
| Payload Receive | Complex | N/A | N/A | 240 |
| Payload Transmit | Complex | N/A | N/A | 240 |
| Thermal Control | Mod Complex | N/A | N/A | 20 |

*Table G.9: Tracking Function Human Operator Times to Completion for Event and Failure Recovery with Level of Automation*

| LOA (Remote, Ground)* | Event Recovery Logged Time (min.) | Event Recovery Elapsed Time (min.) | Fail. Recovery Logged Time (min.) | Fail. Recovery Elapsed Time (min.) |
|---|---|---|---|---|
| (6,6) | 20 | 20 | 5 | 10 |
| (6,5) | 5 | 5 | 5 | 10 |
| (4,5) | 5 | 5 | 10 | 10 |
| (2,5) | 20 | 20 | 30 | 30 |
| (2,4) | 5 | 5 | 25 | 25 |
| (2,2) | 30 | 30 | 30 | 30 |

*LOA 1 = fully automated, 2 = paging, 3 = supervision, 4 = cueing, 5 = data filtering, 6 = no automation

## G.2 Payload Automation of Constellation

All parameters not shown in section G.2 are assumed to be the same as defined in section G.1.

*Table G.10: Payload Automation Marginal Development Cost (relative to baseline)*

| Level of Automation | Flight Software (lines of code) | Ground Software (lines of code) | Marginal Payload Development Cost |
|---|---|---|---|
| No Remote or Ground Automation | 0 | 0 | $0 |
| No Rem. Aut., Ground Data Filter. | 0 | 2500 | $792,500 |
| Remote Cueing, Ground Data Filter. | 12500 | 2500 | $8,617,500 |
| Remote Supervision, Ground Data Filter. | 17500 | 2500 | $10,955,000 |
| Remote Paging, Ground Data Filter. | 17500 | 2500 | $10,955,000 |
| Remote Paging, Ground Cueing | 17500 | 2500 | $10,955,000 |
| Remote Paging, Ground Paging | 17500 | 2500 | $10,955,000 |
| Fully Automated | 18750 | 0 | $11,737,500 |

* Flight Software = $626 per line; Ground Software = $317 per line

*Table G.11: Payload Event and Failure Execution Times*

| LOA (Remote, Ground) | Human Event Recovery Time** | Human Failure Recovery Time** | Flight Software Event Recovery Time | Flight Software Failure Recovery Time | Ground Software Event Recovery Time | Ground Software Failure Recovery Time |
|---|---|---|---|---|---|---|
| (6,6) | 60 | 240 | N/A | N/A | N/A | N/A |
| (6,5) | 30 | 120 | N/A | N/A | N/A | N/A |
| (4,5) | 15 | 60 | 10 | 10 | N/A | N/A |
| (3,5) | 15 | 60 | 10 | 10 | N/A | N/A |
| (2,5) | 60 | 240 | 10 | 10 | N/A | N/A |
| (2,4) | 45 | 180 | 10 | 10 | 5 | 5 |
| (2,2) | 60 | 240 | 10 | 10 | 5 | 5 |
| (1,N/A) | N/A | N/A | 10 | 10 | N/A | N/A |

* LOA's: 1 = Fully Automated, 2 = Paging, 3 = Supervision, 4 = Cueing, 5 = Data Filtering, 6 = No Automation

** Human elapsed time and logged time are identical for this study.

*** All times in minutes.

## G.3 Deep Space Mission

For the deep space mission, both the bus and payload functions are required to meet the mission objective requirements. The bus is assumed to be independent of the payload, and the payload has a type 2 dependency on the bus. The mission life was assumed to be 24 months with a discount rate of 10%. The automation task reliabilities were the same as for section G.1. The operations cost parameters were also assumed to be the same as for section G.1.

*Table G.12: Deep Space Mission Bus and Payload Development Costs*

| Level of Automation | Bus Dev. Costs ($M) | Payload Dev. Costs ($M) |
|---|---|---|
| No Remote or Ground Automation | 75 | 75 |
| No Remote Automation, Ground Data Filtering | 77 | 78 |
| No Remote Automation, Ground Paging | 89 | 98 |
| Remote Cueing, Ground Data Filtering | 112 | 136 |
| Remote Paging, Ground Cueing | 103 | 122 |
| Remote Paging, Ground Paging | 103 | 122 |
| Fully Automated | 101 | 119 |

*Table G.13: Processor Reliabilities for Bus and Payload Event and Failure Recovery*

| | Bus Event | Bus Failure | Payload Event | Payload Failure |
|---|---|---|---|---|
| Space Processor (Cueing or Data Filtering) | 0.999 | 0.99 | 0.99 | 0.9 |
| Space Processor (Paging or Supervision) | 0.9999 | 0.999 | 0.999 | 0.99 |
| Space Processor (Fully Automated) | 0.99999 | 0.9999 | 0.9999 | 0.999 |
| Ground Processor (Cueing or Data Filtering) | 0.999 | 0.99 | 0.99 | 0.9 |
| Ground Processor (Paging or Supervision) | 0.9999 | 0.999 | 0.999 | 0.99 |

*Table G.14: Processor Execution Times for Bus and Payload Event and Failure Recovery*

| | Bus Event | Bus Failure | Payload Event | Payload Failure |
|---|---|---|---|---|
| Space Processor (Cueing or Data Filtering) | 0.25 | 1 | 0.25 | 1 |
| Space Processor (Paging or Supervision) | 0.5 | 2 | 0.5 | 2 |
| Space Processor (Fully Automated) | 0.5 | 2 | 0.5 | 2 |
| Ground Processor (Cueing or Data Filtering) | 0.1 | 0.5 | 0.1 | 0.5 |
| Ground Processor (Paging or Supervision) | 0.2 | 1 | 0.2 | 1 |

*Table G.15: Human Task Execution Times*

| LOA | No Automation | Data Filtering | Cueing | Supervision | Paging |
|---|---|---|---|---|---|
| Bus Event (Rel. = 0.98) | 10 | 5 | 5 | 20 | 50 |
| Bus Failure, Payload Event (Rel. = 0.999) | 15 | 7 | 5 | 30 | 60 |
| Payload Failure (Rel. = 0.999) | 20 | 10 | 5 | 40 | 90 |

* Table G.15 provides the human logged time. The execution time is calculated by the sum of the logged time and the round trip communications time which is dependent on the mission destination.

*Table G.16: Human Supervisory Times for Deep Space Mission*

| Level of Automation | Supervisory Time (persons) |
|---|---|
| Fully Automated | 0 |
| Paging | 0 |
| Supervision | 3 |
| Cueing | 5 |
| Data Filtering | 8 |
| No Automation | 8 |

*Table G.17: Mean Times to Event, Failure, and Permanent Failure*

| | Mean Time to Event (min.) | Mean Time to Failure (min.) | Mean Time to Perm. Failure (min.) |
|---|---|---|---|
| Bus | 720 | 274080 | 5429863 |
| Payload | 60 | 260 | 37733100 |

# Appendix H: Software Model

There exists several approaches to the calculation of software reliabilities [Dhillon]. Most such methods utilize the software's failure rate during code testing to estimate the probability of failure during operation. One such model, the Shooman Model [Dhillon], assumes that software errors are initially at some value. As errors are detected during testing, they are removed, and no new errors are introduced. The number of residual errors is defined as the difference between the number of initial errors and the number of errors found, and the failure rate is assumed to be proportional to the number of residual errors. Therefore, a knowledge of the error rate during testing, along with the number of errors found can lead to a calculation of the expected error rate during operation.

The software model proposed here represents a modification of existing models. Rather than treating the code as continuously running, with errors occurring on a time basis, the errors are assumed to occur on an execution basis, defining probabilities of experiencing an error per execution of the code. This treats the code as a rule-based decision maker. If the input parameters are the same, the rules will produce the same answer. Thus, once a set of inputs have been tested, the code should not fail if the same inputs are experienced. This allows the probability distribution of the inputs to be used to estimate the code's reliability in an actual environment where some inputs are more likely than others.

As with other software models, this software model provides an estimate of the software's reliability based upon software test data. It is based upon the premise that software is first debugged as well as possible, and then tested systematically to uncover deeper bugs. It also assumes that most software is essentially a black box which takes inputs (a state vector). Based upon the values of the inputs, certain sections of the code are executed, and generate outputs. Within this context, testing of the code is performed by feeding the code as many unique input state vectors as practical. Much of the time, all possible combinations of the inputs cannot be tested within a reasonable amount of time. In such cases, those inputs which are most likely to occur are tested, and those which are unlikely