A Framework for Decision Support in Systems Architecting

by

Willard Lennox Simmons

B.S., University of New Hampshire, 1997
M.S., University of Colorado, 1999
M.S.E., Princeton University, 2005

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2008

Author . . . .
Department of Aeronautics and Astronautics
January 11, 2008

Certified by . . . .
Edward F. Crawley
Ford Professor of Engineering
Thesis Supervisor

Certified by . . . .
Brian C. Williams
Professor of Aeronautics and Astronautics
Committee Member

Certified by . . . .
Benjamin H. Y. Koo
Associate Professor of Industrial Engineering, Tsinghua University
Committee Member

Accepted by . . . .
David L. Darmofal
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

A Framework for Decision Support in Systems Architecting

by
Willard Lennox Simmons

Submitted to the Department of Aeronautics and Astronautics
on January 11, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The objective of this thesis is to provide a method and tool to leverage computational resources to empower a systems architect to reason about architectural decisions more comprehensively and effectively compared to traditional approaches. This thesis provides a computational framework for decision support called the Architecture Decision Graph framework. It supports human decision-making by providing a methodology for generating and analyzing architectures as the result of a set of interrelated decisions. ADG's explicit representation of an interconnected decision problem is a bipartite graph of decision variables, property variables, logical constraints, and property functions. The Architecture Decision Graph's framework provide tools for reasoning about the structure of a decision problem, generating the set of feasible combinations of decisions, and simulating their outcome. The underlying computational engine used by ADG is the Object-Process Network (OPN) kernel.

The contribution of this thesis to the field of systems architecting falls into three areas: First, the thesis contributes the ADG representation of an architectural candidate space as a set of interrelated decision variables. Second, the thesis contributes the ADG framework, which leverages the ADG representation of architecture to transform an architecting problem into a computational problem. Third, this thesis contributes decision space viewing tools, which present the potential impact of changes in the assignments of the decision variables to an architect.

The ADG representation, analysis methodology, and tools are demonstrated with two applications. The first application is a retrospective study of the architectural decisions related to the development of the Apollo moon project of the 1960's. The second application is a study of decisions in support of NASA's lunar outpost architecting effort. The applications include discussions of the practical considerations related to the use of ADG as a decision representation method, the efficiency of the simulation algorithm, and a discussion of the architecting insights that can be drawn from the results.

Thesis Supervisor: Edward F. Crawley
Title: Ford Professor of Engineering

Acknowledgments

I would like to thank my committee, Edward F. Crawley, Brian C. Williams, and Benjamin H. Y. Koo for their support of my research. Professor Crawley's enthusiasm for this research was infectious. He introduced me to the field of systems architecting research when I first came to MIT, and I haven't looked back. As an advisor, Professor Crawley was a perfect match for me. He was hands-off for weeks at a time, which allowed me to work on new ideas on my own. Then, when the time was right, we would spend hours at his home on a Saturday, sitting around the dinner table refining the details. I appreciate the confidence he instilled in me. Professor Brian Williams introduced me to the world of autonomous reasoning and Artificial Intelligence research, which is filled with clever ideas for representing and processing knowledge in order to generate new knowledge. Many of the ideas in this thesis had their genesis in his Introduction to Principles of Autonomy and Decision Making. I credit Professor Benjamin H. Y. Koo with changing the way I think about design problems. Over the course of three years, he trained me to look for solutions at the highest level of abstraction first, rather than getting lost in the details.

My two thesis readers, Professor Olivier de Weck and Dr. Jana Schwartz, put a tremendous amount of time and thought into giving me feedback on my thesis. It seems almost unfair that their names are not on the cover of my thesis. Their comments helped me focus the argument for my contribution. Professor Nick Roy was my minor advisor. I'm grateful that he was always willing to set aside the time in his busy schedule to discuss my research.

This achievement would not be possible without the support of my family. Striving for a doctoral degree meant five years of missed dinners, canceled weekend plans, and holiday plans invaded by the clicking sound of a laptop keyboard. When my stress-levels peaked, my wife, Madeline T. Pham, was always there to calm me down. When I was stuck in the office until late at night, she was always willing to pick me up and drive me home. Madeline is a constant source of encouragement and I could not have done this without her. My parents' influence molded me into the person I am today. My father, Theodore R. Simmons, taught me to question everything I am told and everything I have read. Inquisitiveness is the most important trait of a researcher. My mother, Janet L. Simmons, inspired me to overcome bumps in the road of life. Whenever I had a tough day, she always said, Tomorrow's another day. It sounds silly, but it's really true. Tomorrow is another day, and I am there!

My colleagues at MIT have been tremendously helpful. Wilfried Hofstetter took the time read every section of my thesis with his characteristic precision. His feedback was invaluable. The extensive research by Paul Wooster and Wilfried Hofstetter formed the basis for the NASA Lunar Outpost Study in Chapter 5 and helped refine the Apollo study in Chapter 4. Getting to know Ryan Boas has been a pleasure. He was always willing to give professional, sharp, targeted advice on my research. Jaeymung Ahn has been a tremendous help with the details of my work ever since the first day I arrived at MIT. Jaeymung and Lars Blackmore were my study partners for the qualifying exam. Maokai Lin, who arrived at MIT just five months ago, has been helped with many software implementation details. Kathi Cofield, who has been

Ed Crawley's assistant for five years, has always been helpful taking care of scheduling snafus and making sure I get the time I needed with Ed. It's been a lot of fun with her. I can't name all the members of the Space Architects Group who contributed to this effort, but I will say that I was especially moved when about fifteen people showed to my thesis defense practice the Friday before my defense. Five more people came to the office in the middle of blizzard on the following Sunday to help me put the final touches on my talk.

In the past year, I've met two students from Professor Daniel Jackson's lab in EECS. Rob Seater and Derek Rayside, two brilliant computer scientists, have generously spent many hours of their time talking about my research instead of making progress on theirs. I hope I can return the favor soon.

This work was made possible by the generous support of Draper Laboratory under a University IR&D grant. I've thoroughly enjoyed my exchanges with Phil Babcock, Jose Lopez, Jasjit Heckathorn, and Jana Schwartz. Each of them gave me a mixture of encouragement and targeted comments and helped me focus on the key unanswered questions. My semi-annual briefings at Draper have always been able to snap this academic research back to reality.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

The job of a systems architect is to transform a set of needs and goals into an architecture for a system. An *architecture* is a stable, high-level mapping of functions to forms that embodies the concept for a system [Cra07, Koo05]. An architecture does not specify a detailed design, it specifies a category of acceptable designs. For complex systems, the task of architecting is considered challenging because the many-to-many relationships between products and their contexts often introduce a massive search space which challenges either humans' or computers' abilities to exhaustively process the candidate space. Solving difficult search problems often depends on using an effective knowledge representation[BL85]. This thesis argues that a systems architecture search space can be effectively represented as a set of interconnected decision variables.

In this thesis, I assert that systems architecting is fundamentally a decision-making process. For example, consider the process of architecting the Apollo project of the 1960's. In Figure 1-1, President Kennedy is shown giving his famous May 1961 "Urgent National Needs" speech. In that speech, Kennedy stated that the United States must send a man to moon and return him safely to the Earth by the end of the decade in order to achieve the goal of maintaining America's world leadership status [Ken61, Wie95, Sea05]. This set of needs and goals was transformed into an architecture for the system. Systems architects achieved this transformation by making *decisions* to reduce the candidate space of architectures. They *decided* to use lunar orbit rendezvous as a mission-mode. They *decided* to use a single, large launch vehicle. They *decided* to use a crew of three astronauts. By making these decisions, the systems architects provided design engineers with a stable category of acceptable

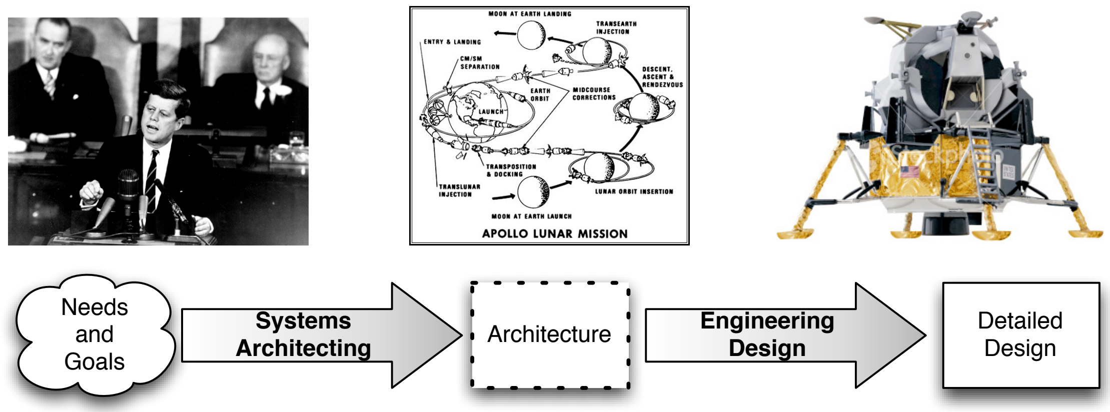


Figure 1-1: Systems architecting transforms a set of needs and goals into an architecture. Engineering design transforms an architecture into a detailed design. (source of images: NASA)

designs that were eventually refined into a single detailed design. The Apollo project successfully completed its goals with the first successful lunar landing in July 1969 [MC04].

Although systems architecting is essentially a decision-making process, representations of architecture generally do not make the relationship between architecture and architectural decisions explicit. In the mathematical sense, *explicit* is defined as "having the dependent variable expressed directly in terms of the independent variables, as $y = 3x + 4$." [dic08]. An explicit representation of architecture as a set of decisions would have architecture as an output of a function of decision variables. Typical architectural representations of systems, such the Object Process Methodology (OPM) [Dor02], the Unified Modeling Language (UML) [OMG03], the Systems Modeling Language (SysML) [Sys06], and the DoD Architecture Framework (DoDAF) [Dep07] do not have this capability. One representation which has can be used to represent architecture as an *implicit* function of decision variables, Object Process Network (OPN) [Koo05], will be discussed in Section 2.3.4. In OPN, architectural decision variables are implicitly represented as a procedural branches in a computational model.

The specific objectives of this research are to develop an explicit representation of architecture as a set of decisions and show that, through using this representation, an architect can gain useful insight into the architectural candidate space. This thesis satisfies these objectives by:

- Introducing Architecture Decision Graph (ADG), an explicit representation of architecture as a set of interrelated decisions.

- Providing a framework which uses the knowledge represented in an ADG to transform an architectural problem into a computational problem.

- Providing a methodology for evaluating the relative degree of potential impact of decision variables.

The remainder of this introduction is structured as follows: The next two sections discuss the definitions of the terms “decision”, “decision support”, “systems architecture” and “systems architecting”. Next, the needs of a systems architect are outlined. The chapter concludes with a synopsis of this thesis.

1.2 Decisions and Decision Support

Various definitions of “decision” can be found in the technical literature. According to R. Hoffman, the word *decide* comes from the Sanskrit word *khidáti*, meaning “to tear”, the Latin word *cædare*, meaning “to kill” or “cut down”, and also the Latin word *decædare*, which means “to cut through thoroughly”[Hof05]. A contemporary English definition of the word *decision* means “the passing of judgment on an issue under consideration” [ah-04]. A third definition for decision-making written by S. Hannson is “goal-directed behavior in the presence of options”[Han05]. The key ideas in these definitions are 1) there exists a controllable situation with multiple alternatives, 2) a selection (or decision) is made which separates (“tears”) the solution space, and 3) there is some expected benefit that will be achieved by making this decision. In this thesis, the word *decision* means *a purposeful selection from mutually exclusive alternatives.* The term *decision-making* is a process which culminates in one or more decisions.

Decision support is the task of assisting decision-makers in making a decision. Before computers were widely available, the idea of decision support largely meant providing a rational[1] procedure for arriving at a decision. According to S. Hansson, formalized methods for decision support can be traced back as far as 1793, when the French philosopher Condorcet outlined a three-phase decision-making process for the French Constitution. “Staged” or “phased” decision-making methods were later

[1]In the context of this thesis, I use the word *rational* to mean, “having or exercising reason, sound judgment, or good sense”[ah-04].

developed by J. Dewey (1910), H. Simon (1960), Witte (1972), and Mintzberg (1976), among others [Han05, Tan06]. Many of these staged processes can be shown to be roughly equivalent to Herbert Simon's four-phase process for decision-making outlined in "The New Science of Management Decision" [Sim60, Sim77]. Simon's breakdown, described below and shown in Figure 1-2, is used as a baseline for this thesis:

1. **Intelligence Activity**: "Searching the environment for conditions calling for a decision."

2. **Design Activity**: "inventing, developing, and analyzing possible courses of action."

3. **Choice Activity**: "selecting a particular choice."

4. **Review Activity**: "assessing past choices."

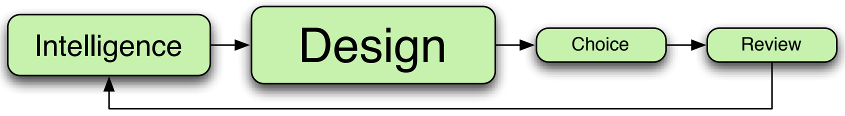


Figure 1-2: H. Simon's four stages of decision-making.

According to Simon, managers (decision-makers) spend a large fraction of their resources in the intelligence activity phase, an even greater fraction of their resources in the design activity phase, and small fractions of their resources in the choice and review activity phases. This breakdown of resources is pictorially indicated by each stage's size in Figure 1-2.

It is important to note that other research has found that the fraction of time spent in these four phases of decision making can be significantly different, depending on the situation and total time available. For example, in the case of high-speed decision-making made by people like military commanders, firefighters, or high-speed chess players, the time spent in the "design activity" for high-speed decision making is very short or virtually non-existant. An excellent study of quick decision-making by people in high-pressure situations with little time to generate alternatives can be found in Klein's "Sources of Power" [Kle99]. In these types of situations, decision-makers tend to draw analogies to previous situations in their experience, and then pick a solution similar to one that worked before. Since the time scale of these decisions is typically on the order of minutes, there is little time to allocate for inventing and

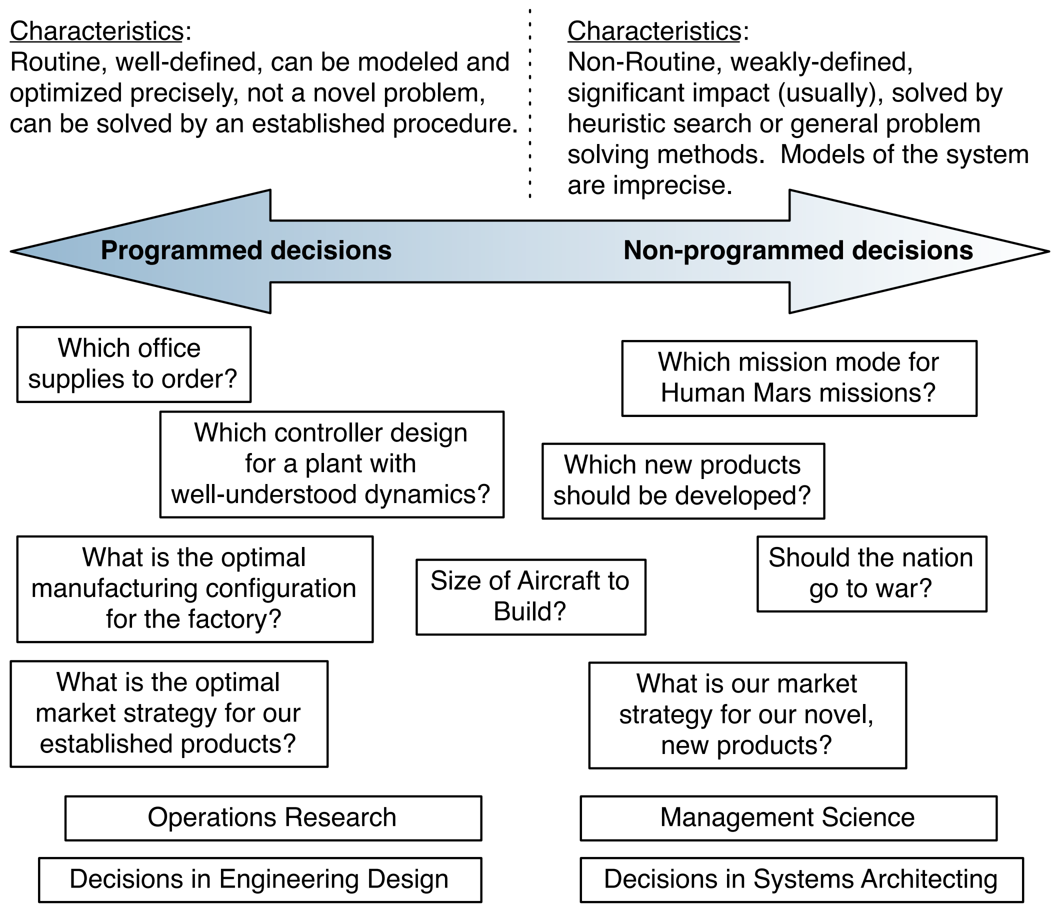

Characteristics:
Routine, well-defined, can be modeled and optimized precisely, not a novel problem, can be solved by an established procedure.
Characteristics:
Non-Routine, weakly-defined, significant impact (usually), solved by heuristic search or general problem solving methods. Models of the system are imprecise.
Programmed decisions
Non-programmed decisions
Which office supplies to order?
Which mission mode for Human Mars missions?
Which controller design for a plant with well-understood dynamics?
Which new products should be developed?
What is the optimal manufacturing configuration for the factory?
Size of Aircraft to Build?
Should the nation go to war?
What is the optimal market strategy for our established products?
What is our market strategy for our novel, new products?
Operations Research
Management Science
Decisions in Engineering Design
Decisions in Systems Architecting

Figure 1-3: The spectrum of decisions: Programmed vs. Non-programmed decisions.

analyzing alternative courses of action. In the case of systems architecture decision-making, the focus of this thesis, the time scale is typically on the order of weeks, months, or years. Therefore, in this thesis, using Herbert Simon's breakdown of the allocation of required resources is appropriate.

Simon's research also makes the complementary observation that there are two "polar" types of decisions: programmed decisions and non-programmed decisions (sometimes called "structured" and "unstructured" decision problems by subsequent authors [TA98].) Examples and characteristics of these two types of decisions are shown in Figure 1-3.

Programed decisions are "repetitive and routine decisions" where a procedure for making decisions for this type of problem has been worked out *a priori*. Examples of programmed decisions range from simple to very complex. For instance, deciding how much to tip a waiter, deciding the optimal gains of a rocket control system, and deciding the routing of all aircraft flying over the United States can all be considered programmed decision problems. In all cases, there is a known and defined approach with which a decision-maker can follow to arrive at a satisfactory choice. Models exist for the behavior of the problem and the objectives are clearly definable. Note that classifying a decision as "programmed" does not imply it is an "easy" decision, it only means that a method to solve it is known and available. The classification of "programmed" does not quantify the amount of resources necessary to use the prepared methodology. For many engineering problems, this pre-determined routine will be difficult to implement or expensive to compute, such as an airline scheduling problem [BLS98]. According to Simon, programmed decisions are considered to lie in the domain of Operations Research.

On the other side of the spectrum are *non-programmed decisions*. These decisions are novel, ill-structured, and often significantly consequential. An example is the decision of the mission mode for the 1960's Apollo mission to land a man on the moon and return him safely to Earth. This decision was regarded as the most difficult and consequential of all of the decisions leading to the successful moon landings [Sea96, Sea05, SKK05]. This problem was extensively studied by NASA and its contractors for two years before a method [Hou61a] to solve this problem was selected. According to Simon, non-programmed decisions are generally solved by creativity, judgement, rules of thumb, and general problem solving methods. Examples are given in Figure 1-3: deciding if a nation should go to war, deciding market strategy for new, unproven products, and deciding the mission mode for human Mars missions. According to Simon, non-programmed decisions are considered to lie in the domain of Management Science.

Many engineers and system architects may balk at this breakdown of programmed and non-programmed decisions. In many cases, decisions thought to be non-programmed become programmed once someone is clever enough to invent a programmed method to solve that problem. Perhaps a better name for non-programmed decisions is "not-yet" programmed decisions. In the following abridged list of four examples, engineers and architects were able to derive a systematic way to program a previously non-programmed problem:

- Wilcox and Wakayama studied the architectural decision of "how should our new aircraft be modularized?" for Boeing's Blended Wing Body concept in Reference [WW03] using a systematic application of operations research techniques.

- Simmons, Koo and Crawley systematically studied mission-modes for the human exploration of the moon and Mars by encoding a motion-language formulation of the problem in Object-Process Networks[SKC05a].

- Fang, Hipel, and Kilgore [FHK93] describe a technique for systematically studying socio-technical-political conflicts using a game theory approach that has been applied to problems such as ground water contamination disputes [HKLP97] and negotiating the Kyoto protocol [WHI07].

- Christopher Alexander introduced the concept of "pattern language"[Ale77, Ale79] as a systematic way to develop civil architectures. The pattern language catalogs elements of an architecture as re-usable triples made up of: the context in which they are relevant, the problem they are trying to solve, and the solution they provide.

From these examples it is evident that there is sometimes an opportunity to "program" a decision problem that was previously thought to be "non-programmed". In each case, a systemic way to analyze the problem space was enabled through an effective representation of the problem. By developing an effective representation, coupled with a methodology to leverage its information, a system architect can comprehensively and efficiently examine a solution space using rigorous analysis rather than using heuristics to pre-simplify the space, as suggested by Simon and others [MR02].

1.3 Systems Architecture and Systems Architecting

The term *systems architecture* is defined by B. Koo as "the stable properties of a system of interest"[Koo05]. A systems architecture is "stable" in the sense that, at some particular level of abstraction, the description of the behavioral and structural properties of the system does not change. E. Crawley defines architecture as "the embodiment of a concept: the allocation of physical/informational function to elements of form, and the definition of interfaces among the elements and with the surrounding context"[Cra07].

Systems architecting is the process of creating a systems architecture[CdWE^{+}04, Rec91, MR02]. A systems architect generally starts with a vague set of needs and goals and a definition of the context in which the system will operate. This set of information provides an initial boundary for possible implementations of the system.

As architecting proceeds two things happen: 1) additional information about the system and its context is gained and 2) decisions are made about its implementation. Both types of inputs to the systems architecting process have the possibility of widening or narrowing the space of possible implementations. For the purposes of this thesis, I define *systems architecting* as the process of transforming needs, goals, and context into a systems architecture, as opposed to *engineering design*, which is the process of transforming a system concept into an implementable design.

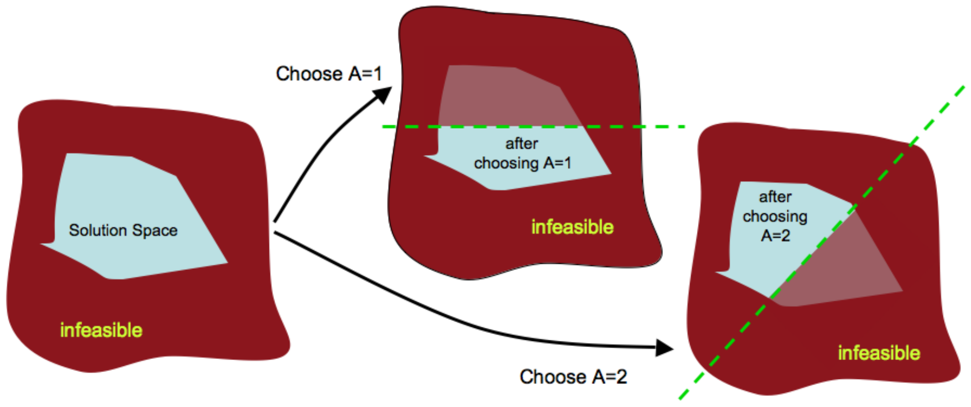


Figure 1-4: This notional architecture solution space would be changed by a decision about the systems architecture.

Figure 1-4 is a notional depiction of the idea that the steps of systems architecting can be characterized by decision-making. The set of all possible systems that could satisfy the needs and goals in a particular context can be represented by a hyper-space of discrete points. Each point in this hyper-space represents a feasible system concept which could satisfy all of the needs and goals to some acceptable extent. In this hyper-space of system concepts, the feasible solution space is bounded by the needs and goals, as well as the context in which it operates (this includes considerations such as obeying the laws of physics). In this example, the solution space is narrowed after making a decision about decision variable "A", which could be assigned the value of either "1" or "2". The feasible solution space is changed in a different way by selecting one alternative over the other.

In Systems Architecting, a *decision* can be thought of as a partitioning and selection operation in the architectural candidate space. When a decision is made the system architect chooses a part of the solution space that will remain in consideration, and a part that will be eliminated. In practical terms, the architect reduces the set of alternative implementations that are acceptable.

A concrete example of this concept is shown in Figure 1-5. This Object Process Method (OPM) [Dor02] diagram is a simplified description of a system architecture for a traveller who wants to go from New York to Los Angeles given constraints on Time Available, Budget, and Comfort Needs. The architect (possibly the traveller himself) is presented with a choice between transportation systems, indicated by the OPM specialization symbol, △. The specialization symbol indicates that the transportation system can be specialized to be a Car, Plane, or Train. Each of these transportation systems has its own associated Speed of Travel, Cost, and Comfort level.

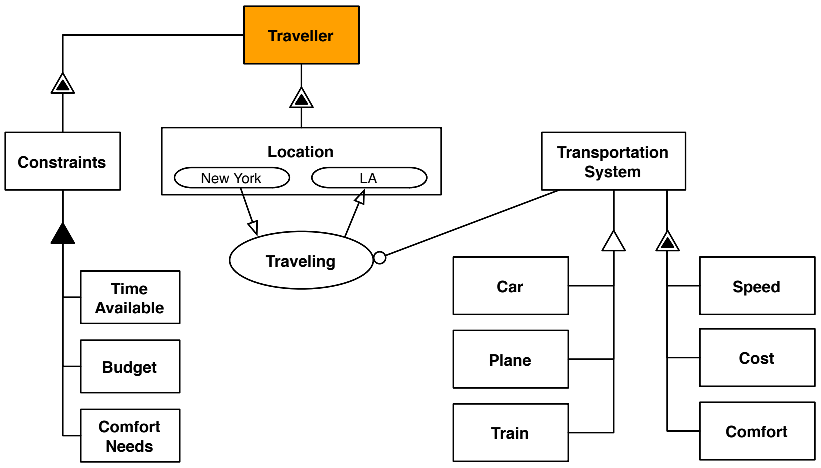


Figure 1-5: A simple architecture for a traveller who desires to go from New York to Los Angeles. (This diagram is in Object-Process Methodology (OPM) notation [Dor02].)

In a typical large system, decisions are not as simple as the one depicted in Figure 1-5. Commonly, there are many decisions that are interrelated in unclear ways. As an illustrative example, consider the extensive research conducted at MIT from 2004 to 2006 on the architecture for implementing the Vision for Space Exploration (See References [SKC06, SKC05b, CCC06, HWNC05, WHC05a, BAH$^+$05, HWSC06, WHC05b]). This research illustrates that that the mission-mode decision for human moon and Mars exploration missions is connected to the launch vehicle decision, the spacecraft partitioning decision, decisions about the distribution of the NASA workforce in the next ten years, decisions about who wins subcontractor awards, and decisions about scientific goals, among others. Another illustrative example is the systems architecting of a novel arctic oil exploration system[Roz07]. A company aiming to build an arctic oil exploration facility must consider the interactions between decisions about local and international political strategy, environmental issues, power supply & storage, ice protection, and seismic protection.

1.4 Needs of the Architectural Decision-Maker

Systems architecting can be viewed as a decision-making process for non-programmed decision problems. Therefore, Simon's stages of the decision-making process also form the partitions of the basic needs of a systems architecture decision maker: Intelligence, Design, Choice, and Review. An architectural decision-maker needs effective methods for each of the four stages of the decision-making process.

For the *intelligence activity*, a decision maker needs effective methods for identifying occasions for making a decision and for understanding the value proposition of the decision-making environment. For the *design activity*, a decision-maker needs methods for generating and analyzing feasible sets of alternatives. For the *choice activity*, a decision-maker needs methods for selecting courses of action while taking into consideration the predicted impact of each generated alternative. Since decisions in organizations are updated on a continuing basis, there is also a *review activity*, where a decision-maker needs methods for assessing past choices and using that information to revise assumptions and models of the decision problem. In principle, the review activity could be considered as revision of the intelligence activity.

According to Simon, the most time-consuming activity in decision-making for non-programmed problems is the design activity. Even if the intelligence, choice, and review activities have perfect solutions, the design activity is still potentially the most difficult stage of the decision-making process. Designing and analyzing the set of feasible alternatives in systems architecting is considered *hard* because it often introduces a search space that involves divergent or infinite possibilities. Searching in a divergent space of possibilities is not only hard for computers, it is also hard for humans to construct a consistent and stable mental model.

Because the design activity is the most-resource intensive process, the methodology in this thesis focuses on the needs of a systems architect in this specific area. The needs of a systems architect, in terms of design activity, can be broken down into further detail. A survey of decision support literature [Pow02, BHW81, TAL05], reveals that the task of the design activity can be represented as four "layers": representing, structuring, simulation, and viewing:

The *representing* layer includes the methods and tools for representing the problem for both the human decision-maker as well as representing the problem using an encoding that a computer can interpret.

The *structuring* layer refers to the methods and tools for reasoning about the structure of the decision problem itself. This includes determining the order that decisions should be addressed or the degree of connectivity between different decision variables.

The *simulating* layer is used to determine which combinations of decisions will satisfy logical constraints and calculate the properties of predicted decision outcomes using metrics.

The *viewing* layer is the methods and tools for presenting decision support information derived from the structuring or simulating layers in a human-understandable format. In some literature, the viewing layer is combined with the representing layer, since it is another form of representing knowledge.

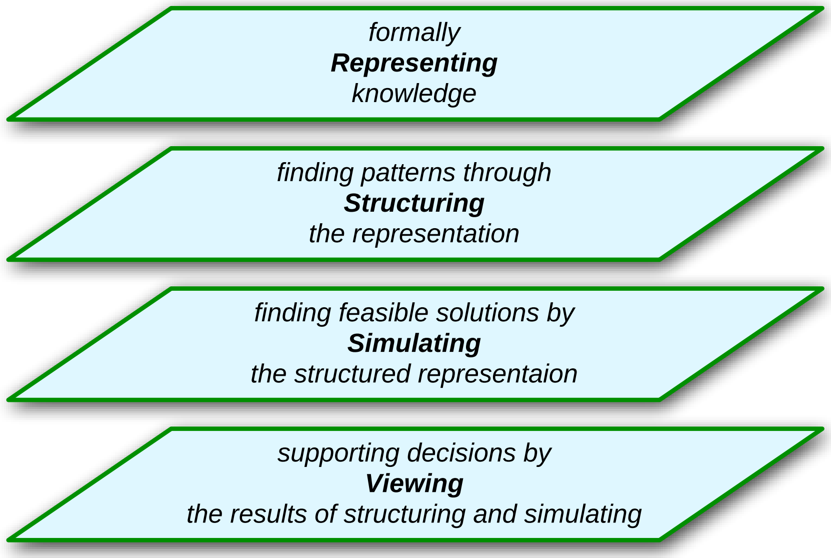


Figure 1-6: The four layers of design activity needs in support of architectural decision-making.

These four layers, shown in Figure 1-6, are used throughout this thesis as a way to modularize the decision support problem.

1.5 Summary and Synopsis

The intent of this thesis is to provide a framework for decision support that satisfies the representing, structuring, simulation, and viewing needs of an architectural decision maker. Specifically, the goal of this thesis is to support the "design" activity phase for "non-programmed" decision problems by providing a decision support framwork. The four examples presented in Section 1.2 show that by providing a framework for decision support in systems architecting problems, the non-programmed decision problems can move to the left on the "spectrum" and become more "programmed". This thesis provides tools for inventing, developing and analyzing possible courses of action for novel architecting problems so that they can be analyzed more effectively and efficiently.

The remainder of the thesis is structured as follows:

Chapter 2 provides an overview of the state of practice in decision support systems. Existing decision support methods are compared with the four layers of needs for the design activity of architectural decision support.

Chapter 3 provides a new computational framework for decision support called Architecture Decision Graph (ADG). ADG represents an interconnected decision problem as a bi-partite graph of decision variables, property variables, logical constraints, and property functions. ADG's algorithms provide tools for reasoning about the structure of decision problem and simulating the outcome of all sets of feasible combinations of decisions. Information about the sets of feasible combinations of decisions are viewed using two methods: Pareto front plots and decision space views.

Chapter 4 and 5 demonstrate ADG's representation, methodology and tools on two aerospace applications. The first application, Chapter 4, is a retrospective study of the Apollo architecture problem from the 1960's. The second application, Chapter 5, is a study of human lunar outpost architectures for an on-going study sponsored by NASA headquarters.

Chapter 6 contains conclusions, a list of contributions, and a discussion of future work.

2

Literature Review

2.1 Overview

Chapter 1 established the high-level terms and concepts used in this thesis and outlined the needs of an architectural decision-maker. This chapter defines decision support systems and reviews their state of practice. As pointed out in Section 1.4, in the case of architectural decision-making, the design activity is most time-consuming stage of H. Simon's four stage model of decision-making. The design activity involves "inventing, developing, and analyzing possible courses of action" [Sim77]. Since the design activity is the focus of this research, the specific goal of this chapter is evaluate the state of practice in decision support tools and to evaluate them against the representing, structural reasoning, simulating, and viewing needs of architectural decision-support.

2.2 Definition of Decision Support System

As stated in Chapter 1, a *decision* is a purposeful selection from mutually exclusive alternatives. *Decision support* is the task of assisting decision-makers in making a decision. Before computers were widely available, the idea of decision support largely meant providing a rational procedure for arriving at a decision. Today, many software programs provide decision support systems (DSS) by implementing some form of automation to support a rational decision making process. These computer software programs generally assist decision-makers by enhancing communication and processing information to produce new knowledge in support a decision-making process. D. Power refers to different decision support systems as communication-driven, data-driven, document-driven, knowledge-driven, and model driven [Pow02]. The goal of these systems is to enhance the efficiency of decision-makers by providing tools to quantitatively and qualitatively explore and increase information about a space of alternatives for single or multiple decisions.

The definition of DSS is generally wide-reaching; nearly any kind of software that is used to organize and increase the amount of useful information available to a decision-maker could be included within this definition. Taken to the extreme, a word processor document where decision data is recorded, simple spreadsheets which compute financial information, or presentation slides which provide a more succinct presentation of information to a group may be included under the umbrella of decision support systems [Alt80]. For the purposes of this research, the desirable aspects of an architectural decision support system are more narrowly defined in the next section.

2.2.1 Aspects of an Architectural DSS

The goal of this thesis is to provide a decision support system that satisfies the representation, structural reasoning, simulation, and viewing needs of an architectural decision-maker during the “design activity” stage of decision-making. (See Section 1.4). In order to compare the state of practice in decision support tools against these specific needs, the following four desirable aspects of an architectural decision support system are defined:

- **Representational Aspect** – Methods and tools for explicitly (see Section 1.1) encoding architectural decision variables with a set of alternatives, a set of constraints between decision variables, and a set of metrics to calculate system properties.

- **Structural Reasoning Aspect** – Methods and tools for analyzing the structure of interconnected decision variables. Includes providing information about the connections between decisions and the ability to manipulate the structure of the decision problem in order to provide for more efficient simulation of decisions.

- **Simulation Aspect** – Methods and tools for the identification of feasible combinations of decision variable assignments and the computation of multiple properties of those feasible combinations through the application of metrics.

- **Viewing Aspect** – Methods and tools for the presentation of decision support data from either structural reasoning or simulation results in a human-understandable format, such as visual plots or tables.

The term "aspect," which is sometimes called a "cross-cutting concern", is used primarily by computing scientists to describe parts of a program that are not easily partitioned into separate modules [KLM+97, GSS+06]. For example, the representational aspect, the structural reasoning aspect, and the simulation aspect are cross-cutting, because the structural reasoning aspect is enabled by the specification of the representation and the efficiency of the simulation aspect may be dependent on the structural reasoning techniques. The viewing aspect is cross-cutting because it involves presenting structural reasoning and simulation information to the decision-maker. Since the details of these four parts of the problem are interwoven and not easily partitioned into independent software modules, the four parts of the problem are considered aspects.

The following sections discuss the state of practice in decision support in terms of the four aspects given above. At end of each subsection, each of the decision-support methods or tools is evaluated against how well it provides the representational, structural reasoning, simulation, and viewing aspects of architectural decision support. The state of practice section is followed by a summary of the literature review.

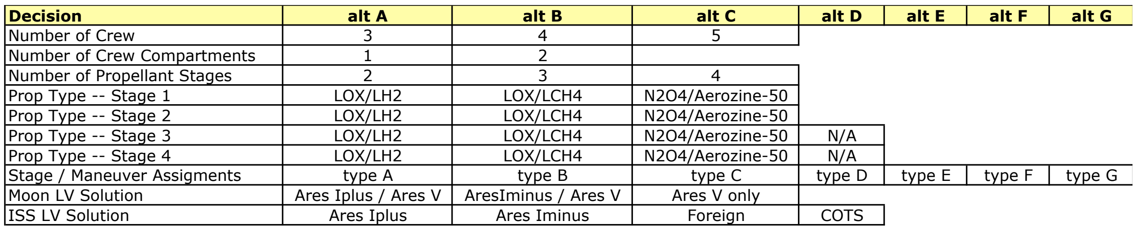
2.3 State of Practice

2.3.1 Table and Matrix-Based Decision Support

Morphological Matrix

The Morphological Matrix is a common way to organize decisions in a tabular format. The morphological matrix was first introduced by Fritz Zwicky in the 1950's as a part of the morphological method for studying the "total space of configurations" (morphologies) of a system [Zwi66]. The use of morphological matrices as a decision support tool is promoted in the engineering design text book by Pahl and Beitz [PB95], among others [Rit06]. Figure 2-1 shows an example of a morphological matrix for decisions related to the design of a lunar lander vehicle (see Reference [SKC06]).

A configuration of the system is chosen by selecting one alternative (labeled "alt") from the row of alternatives listed to the right of each decision. Note that alternatives



Decision	alt A	alt B	alt C	alt D	alt E	alt F	alt G
Number of Crew	3	4	5				
Number of Crew Compartments	1	2					
Number of Propellant Stages	2	3	4				
Prop Type -- Stage 1	LOX/LH2	LOX/LCH4	N2O4/Aerozine-50				
Prop Type -- Stage 2	LOX/LH2	LOX/LCH4	N2O4/Aerozine-50				
Prop Type -- Stage 3	LOX/LH2	LOX/LCH4	N2O4/Aerozine-50	N/A			
Prop Type -- Stage 4	LOX/LH2	LOX/LCH4	N2O4/Aerozine-50	N/A			
Stage / Maneuver Assigments	type A	type B	type C	type D	type E	type F	type G
Moon LV Solution	Ares Iplus / Ares V	AresIminus / Ares V	Ares V only				
ISS LV Solution	Ares Iplus	Ares Iminus	Foreign	COTS			

Figure 2-1: An example morphological matrix for lunar landers.

for different decisions in one configuration of the system do not have to come from the same column. For example, a configuration of this system could be: number of crew = 3 (alt A), number of crew compartments = 2 (alt B), number of propellant stages = 4 (alt C), C propellant type = LOX/LH2 (alt A), and so forth.

In terms of decision support, the morphological matrix is a useful, straightforward method for representing decision variables and alternatives. It is easy to construct and simple to understand. However, since a morphological matrix does not represent metrics or constraints between decisions, it does not provide tools for structuring a decision problem or simulating the outcome of decisions. As a result of not providing structural reasoning or simulation features, morphological matrix also does not provide a viewing aspect.

Although it does not meet the criteria for structural reasoning, simulation, and viewing listed in Section 2.2.1, the concept of a morphological matrix becomes a useful part of the representing and viewing aspects of the approach presented in Chapter 3 by providing a simple representation of decision variables and alternatives for a decision problem.

Design Structure Matrix

DSM is an acronym which generally stands for *Design Structure Matrix*[EWSG94, dsm], but sometimes is written as Decision Structure Matrix or Dependency Structure Matrix. A DSM is an n-squared matrix that is used to map the connections between one element of a system to others. It's called n-squared because it is constructed with n rows and n columns, where n is the number of elements.

When a DSM is used to study the interconnections of decisions, each row and column correspond to one of the n decisions. Figure 2-2 is an example DSM for the lunar lander decisions listed in the morphological matrix in Figure 2-1. This DSM is partitioned and sorted according to the rules in References [Ste65] and [Ste81].

The number 1 in the intersection between "Number of Crew Compartments" and "Stage / Maneuver Assignments" indicates there is a connection between these two decisions. A blank entry in the intersection indicates that there is no direct connection between these decision variables. The definition of a "connection" could be different depending on the specific use of the DSM. For example, it could mean that there exists a logical constraint between the variables, or that a metric function depends on those two variables. In some cases, the entries in the matrix may have different letters, numbers, symbols, or colors in them to indicate different types of connections.

Name		1	2	10	3	8	9	4	5	6	7
Number of Crew	1	1	1								
Number of Crew Compartments	2	1	2								
ISS LV Solution	10			10							
Number of Propellant Stages	3	1			3	1					
Stage / Maneuver Assigments	8				1	8					
Moon LV Solution	9			1			9				
Prop Type -- Stage 1	4						1	4			
Prop Type -- Stage 2	5						1	1	5		
Prop Type -- Stage 3	6	1					1		1	6	
Prop Type -- Stage 4	7	1					1			1	7

Figure 2-2: A sorted, partitioned DSM for lunar landers

By itself, a DSM provides a representation layer and structuring layer for decision support. It represents the decision variables and their interconnections, however, it does not represent the alternatives for each decision or metric calculation functions. Using structuring algorithms, a DSM can be sorted to show which sets of decisions are tightly coupled and which ones are uncoupled. A common use of a DSM is to break a problem into subproblems for different design teams by identifying clusters of closely related decisions and separating them from other clusters of closely related decisions. Since DSM is focused primarily on the connections between decisions and does not provide a way to record the outcome of choosing different alternatives, it does not provide a simulation aspect which meets the requirements of a DSS for architectural decision support as listed in Section 2.2.1. One of DSM's strengths is that its matrix representation can be concurrently used as a way to view structural reasoning results.

2.3.2 Tree and Directed Graph-Based Decision Support

Decision Tree

A *Decision tree* [Rai68] is a well-known way to represent sequential, connected decisions. An example of a decision tree is shown in Figure 2-3. In general, a decision tree has three types of nodes: chance nodes, decision nodes, and leaf nodes. Decision nodes represent decision variables, which are controllable by the decision maker and have a finite number of possible assignments represented by branches in the tree. Chance nodes represent chance variables, which are not controllable by the decision-maker and also have a finite number of possible assignments, which are also represented by branches. The endpoints or "leaf" nodes in the decision tree represent a complete assignment of all chance and decision variables.

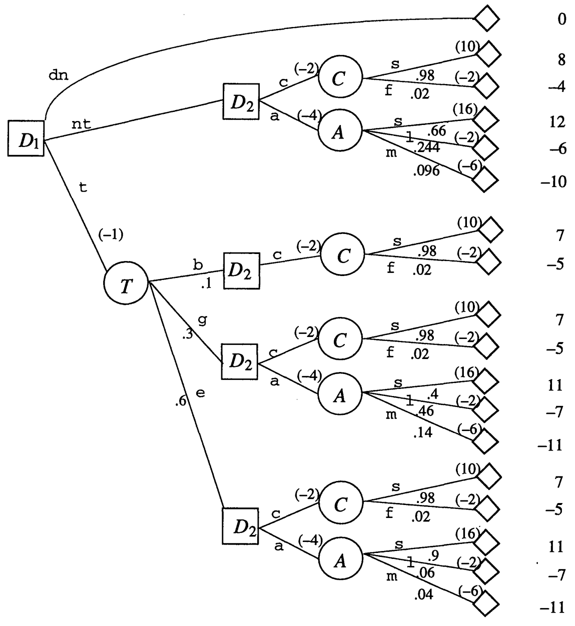


Figure 2-3: Decision Tree from [CO95]. In this tree, decision nodes are squares, chance nodes are circles, and leaf nodes are diamonds.

Covaliu [CO95] and Kirkwood [Kir93] point out some positive and negative characteristics of decision trees. A decision tree is an explicit representation of the entire configuration space for an interconnected decision problem that is easily understood by decision-makers. Simulation of sets of decisions and chance outcomes that are modeled in a decision tree is achieved by calculating the expected utility for each leaf node. This is done using a recursive algorithm. An example of the algorithm can be found in Reference [Kir93].

A key assumption in a decision tree is that a decision problem is sequential. A decision or chance outcome leads to a subsequent decision or chance outcome in a pre-specified order. The implication is that succeeding variables are dependent on the preceding variables. The notions of parallel decision making or conditional independence of decisions are not explicitly captured in a decision tree.

An additional issue with a decision tree representation is that its size grows exponentially with the number of nodes in the problem. It can be extremely large even for moderately sized problems [Kir93]. A decision tree with 10 decisions, each with 3 choices leads to $3^{10} = 59049$ endpoints. Ten is not considered a large number of decisions for many engineering or business decision problems. As a result of decision tree's representational inefficiency, structural reasoning is available only on a limited basis. In order to investigate the possible implicit conditional independence of one decision variable on another one, it is necessary to compute across all branches of the tree that include that variable.

The decision tree formulation for sequential decision problems can be enhanced using the influence diagram (ID) and the sequential decision diagram (SDD). These two enhancements for decision tree are discussed in the following subsections. After the presentation of ID and SDD, the applicability of these three related methods to the design activity for decision support in systems architecting will be discussed.

Influence Diagram

An *influence diagram* (ID) [CO95, HM84, HM05] can be thought of a more compact representation of the variables and structure of a decision problem than a decision tree. An example of an influence diagram is shown in Figure 2-4. Like the decision tree in Figure 2-3, decision nodes are squares and chance nodes are circles. Conditional independence between variables and information flow is indicated using directed arcs. If a variable depends on the input from another variable, an arc is connected from the preceding variable to the succeeding variable. Unlike a decision tree, an influence diagram is representationally space efficient; the size of an influence diagram grows linearly with the number of variables. Each variable is indicated in the ID once, including the multiple leaf nodes depicted in the decision tree, which are compressed to one "sink" node (marked with V in the figure).

Influence diagrams are considered a useful representation because they explicitly capture the information that some variables are irrelevant given certain assignments to other variables. An irrelevant decision variable is one that is not applicable given the results of other variables. This information can be used to pre-compile a corresponding decision tree to reduce the total number of branches. The structure of the influence diagram directly impacts the decision tree construction. In the case of example in Figure 2-4, one possible decision tree realization is given in Figure 2-3.

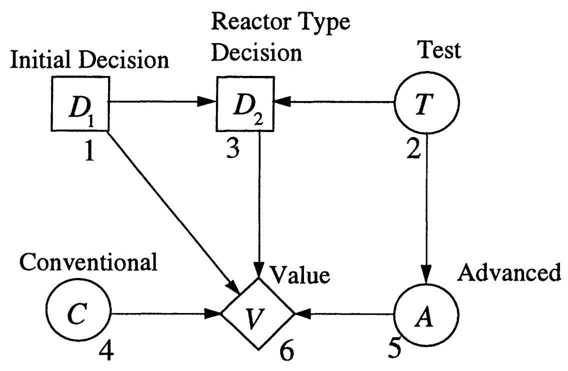


Figure 2-4: An Influence Diagram from [CO95]

Notice that some branches in the decision tree are skipped if they are irrelevant. This is a key advantage since the pre-compilation can eliminate potentially wasted computing cycles.

Like decision trees, influence diagrams assume there is a fixed sequence of influence in a decision problem. However, influence diagrams provide more structural reasoning ability than a decision tree alone, since variables that do not directly influence each other can be identified by following paths through the diagram. When an influence diagram is compiled into a decision tree, the variables in the tree can be arranged in any sequence as long as all successor nodes always follow their predecessor nodes in the ordering. Using an influence diagram to guide the construction of a tree allows the decision tree to be built in such a way that it is more space and time efficient [HM05]. Structural reasoning is provided by the ability to sequence a decision tree before simulating it. Since simulation of IDs are provided by decision tree algorithms, it is limited to expected utility calculation. Influence diagrams provide one form of viewing. They can be used to generate a visualization of a corresponding decision tree. However, as pointed out above, the decision tree may be impractically large if there are a moderately large number of variables involved.

Sequential Decision Diagram

Covaliu and Oliver present an alternative to decision trees and influence diagrams called *Sequential Decision Diagrams* (SDD) [CO95]. SDD is a directed acyclic graph that is similar to an influence diagram. It contains more specific information than an influence diagram, such as variable assignments and rules for feasible and infeasible combinations of decision assignments. The directed arcs in a SDD specify both the possible assignments of a variable and the rules that may prevent a combination of alternatives of particular variable assignments. Similar to decision trees and influence diagrams, sequential decision diagrams have the limiting assumption that there is a particular sequence of decision and chance variables that must be followed for a decision problem.

A SDD is shown in Figure 2-5, which corresponds to the same decision problem as the decision tree in Figure 2-3 and the influence diagram in Figure 2-4. In this example, the decision variable D_2 can lead to chance variables A or C; however, D_2 will only lead to A if $T \neq b$. Like an influence diagram, the multiple leaf nodes of a decision tree are compressed into one "sink" node called V.

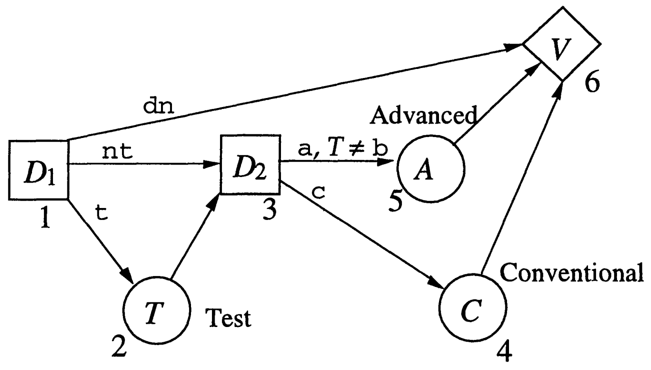


Figure 2-5: Sequential Decision Diagram from [CO95]

SDD provides a more comprehensive representation of a decision problem than an influence diagram since it includes alternatives in the representation. Alternatives are shown as annotations on the directed arcs. Furthermore, decisions or chance nodes that are only relevant if a certain decision is made are also indicated in the SDD. In Figure 2-5 the first decision in D_1. The set of alternatives for D_1 are `t`, `nt`, and `dn`. If t is selected, then the chance node for T is relevant. Likewise, if `nt` is selected, then the decision node for D_2 is relevant. Finally, if `dn` is selected, then all other nodes are not relevant. These implications of the SDD sequence in Figure 2-5 are clearly shown in the Decision Tree in Figure 2-3. By pre-specifying the sequence before creating the decision tree, branches that are irrelevant can be eliminated.

Sequential Decision Diagrams provide some forms of structural reasoning. Since some (but not necessarily all) of the alternatives for decision variables are explicitly shown in the SDD, a decision-maker can draw some conclusions about the connectivity between decision alternatives and other decision or chance nodes, when the relevant information is presented in the diagram. However, since the sequence of the SDD is fixed by the decision-maker who enters the problem, the notion of manipulating the structure of the decision problem is not provided by SDD.

According to Reference [CO95], SDD's algorithms for decision simulation is more efficient than a traditional decision tree. However, the type of information is limited to the calculation of a linear, monotonic utility function that is the sum of the products of utilities and probabilities. The calculation of multiple value functions or non-linear functions of the decision is not possible using neither SDD, ID, or decision trees.

Evaluation of the Decision Tree, Influence Diagram and Sequential Decision Diagram

Among the three preceding representations presented in this subsection, decision tree, influence diagram, and sequential decision diagram, the sequential decision diagram is strictly better than the other two in terms of representing, structural reasoning, and simulating. In terms of viewing the structural reasoning results, SDD and Influence diagrams provide the same type of viewing results: they both have the ability to generate a visual decision tree. In terms of the specific needs for architectural decision support, these methods are not directly used by the framework presented in Chapter 3 for two reasons: they assume that the evaluation of the decision variables is a sequential process. In Chapter 3 we show that architectural concept design problems are more appropriately represented as non-sequential decision-problems rather than sequential decision problems.

Markov Decision Process (MDP)

A Markov model, or *Markov decision process* (MDP), is a well known encoding for structured decision processes ("programmed" decisions in Simon's terminology) which contain both uncontrollable, probabilistic variables and controlled variables. A Markov decision process is formally written as the 4-tuple, $\langle S, A, P_a, g_a \rangle$, where S is a state space, A_x is a set of actions possible at state x, $P_a(x, y)$ is the probability of transitioning from state x to state y, and $g_a(x)$ is the reward at state x when action a is taken[Puc06, Ber05].

MDP's are characterized by the *Markov assumption* (sometimes called the Markov property), which assumes that the probability distribution of future states only depends on the current state. In other words, the state history is not taken into account when calculating the probabilities or rewards for the next state. MDPs are generally solved using dynamic programming techniques, such as value iteration.

There are many uses for MDPs. Typical problems include the so-called "knapsack" optimal packing problem, which determines the way to fit the most items into one or more containers (useful in the logistics domain). Another common MDP problem is the financial portfolio selection problem with discounted returns [Puc06]. MDPs have even been used to combinatorially create music [Roa96]. S. Brin and L. Page used an MDP over a graph representation of the world wide web to create Google's famous PageRank^{TM} system [BP98]. Markov models can be extended to include partially observable states and observations, to become partially observable Markov decision processes (POMDP) or hidden Markov models (HMM). One common application of these methods is speech recognition [Rab89, RJ86, Ber05, Mon82].

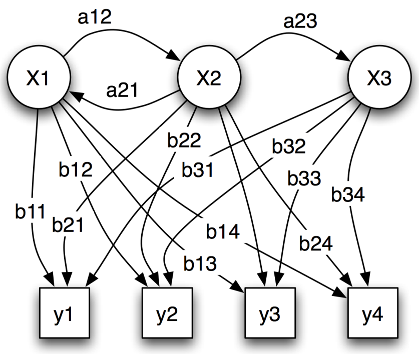


Figure 2-6: A Hidden Markov Model. x represents states, y represents observations, a represents actions, and b represents output probabilities. Source: [Wik07].

Although there are many excellent uses for MDPs in engineering, it is best suited for programmed (structured) decision problems where the possible set of states, actions, and transition probabilities can be derived before any computation takes place. For structured decision problems, Markov models have been shown to be an effective representation and there exist many different simulation methods to find optimal solutions. Some of these tools may exploit the structure of the problem to speed up computation. However, for an unprogrammed/unstructured problem, like decisions in systems architecting, an MDP problem would be difficult to use. For instance, the possible configurations of the system are generally unknown at the beginning of the decision making process. Another issue is that in systems architecting, future decisions are dependent on multiple past decisions. In general, the Markov assumption will not hold. However, in the some methods, such as the Time-Expanded Decision

Network (TDN), the Markov assumption can be side-stepped by including the state history in the state itself.

Time-Expanded Decision Network (TDN)

One application of Markov models to support decision-making in systems design is Silver and de Weck's Time-Expanded Decision Network (TDN) [SdW07]. TDN avoids some of the limitations of a Markov model by including a state history as part of the definition of the state. Although this strategy is usually avoided in the formulation of a Markov model because it negatively impacts computational performance [Ber05], it isn't a major concern in this case because TDN models are always acyclic (no infinite loops) and have a relatively small number of nodes. Figure 2-7 is an example visualization of a TDN.

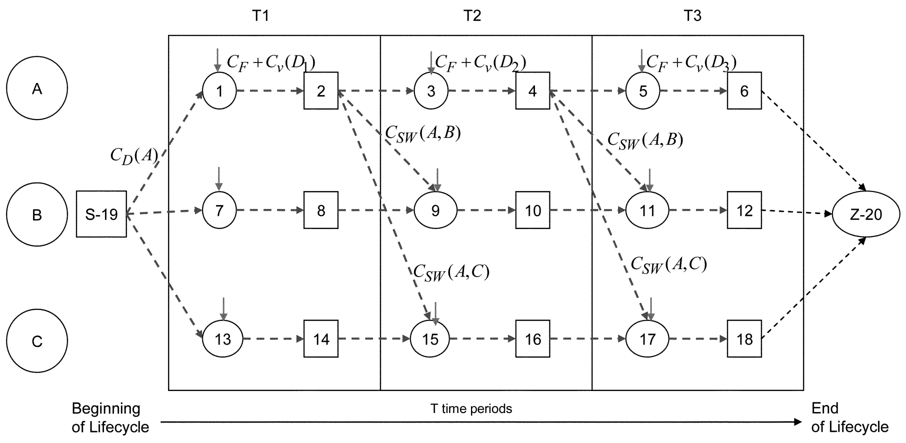


Figure 2-7: An example of a Time-Expanded Decision Network (TDN). Operating and switching costs over time are modeled for three concepts, A, B, and C. (source [SdW07])

TDN is used as a way to generate an optimal strategy for switching between design concepts depending on uncertain future. It can be used as a design tool to quantify the value of real options in terms of the effect of reducing switching costs [de 90]. As a representational method, TDN can show the relationship between different switching decisions between architectural concepts. TDN does not provide structural reasoning tools, since its structure is assumed to be a static time-expanded network. TDN's simulation tools can be used to calculated the expected cost of switching between architectural concept as well as identify paths through the network which are more robust to operating conditions changing over time. As a viewing tool, TDN provides methods presenting the information about the switching costs.

Although TDN is a powerful tool for its indented application, it is based on different assumptions about the architectural space than the ones used in this thesis (assumptions will be listed in Chapter 3). TDN assumes that a selection of architectural concepts are available at the beginning of the decision analysis. The approach presented in Chapter 3, Architecture Decision Graph (ADG), does not assume this is the case. ADG assumes the process starts in the design-activity stage of decision-making, and the possible courses of action must be invented first, then analyzed. The differences between TDN and ADG provide an opportunity for future research, which will be discussed in Chapter 6.

2.3.3 Constraint Graph-Based Decision Support

Constraint Models (CSP and SAT)

Constraint Satisfaction Problems (CSP) are characterized by a set of decision variables each with a finite domain of possible assignments. Decision variables are interconnected by constraints. CSP problems can be represented visually as a graph, where the nodes indicate decision variables and the arcs indicate the constraints between variables. Propositional satisfiability problems (also known as SAT problems) are CSP problems where the decision variables are limited to the values true and false.

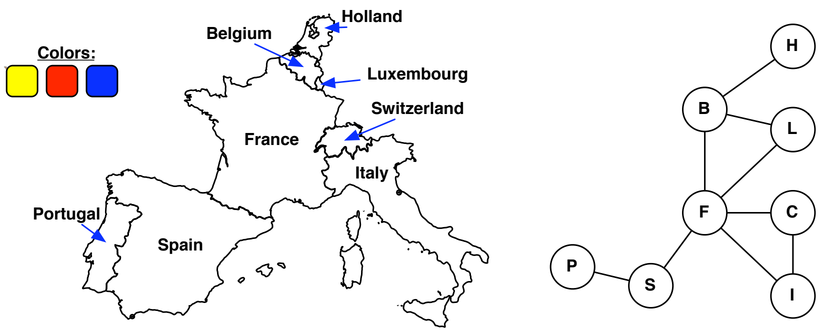


Figure 2-8: An example of a map-coloring constraint satisfaction problem. On the left, a map of western Europe that must be painted using the three colors, yellow, red, and blue such that no two adjacent countries have the same color. On the right, the visualization of the corresponding constraint satisfaction problem representation.

Standard CSP problems only allow "hard" constraints. Every hard constraint specified between variables must be satisfied to achieve a valid solution. An example of a standard CSP decision problem is shown in Figure 2-8. This is a typical map-coloring problem. The partial map of Europe includes eight countries which must be painted with the colors yellow, red, or blue. The constraints in this problem (represented by arcs in the graph on the left) state that no two adjacent countries can have the same color.

A valued-CSP is a CSP that is augmented with a global cost function. The functional elements that make up the global cost function are often called "soft" constraints. Soft constraints indicate the "weight" for particular decision variable assignments. A Valued CSP can be written as a 4-tuple $\langle X, D, C_h, C_s \rangle$, where

- $X = \{x_1, x_2, ..., x_n\}$ are decision variables.
- $D = \{d_1, d_2, ..., d_n\}$ are the decision variables' domains (d_i is a finite set of assignments for variable x_i.).
- $C_h = \{H_1, H_2, ..., H_k\}$ is a set of k hard constraints, and is defined by a scope of variables it applies to, and a logical relationship among the variables.
- $C_s = \{F_1, ..., F_m\}$ is a set of m soft constraints, which are the functions F_j. The functions are summed together to form a global fitness metric, $M = \sum_{j=1}^{m} F_j$.

An example of a valued-CSP decision problem is the map-coloring problem from Figure 2-8 augmented with a function that adds costs for each color of paint. The cost function is implemented using soft constraints that measure the cost for each paint color. For example, the soft constraint might specify that yellow paint costs \$2, red paint costs \$4, and blue paint costs \$6. The global objective is to find a solution to the map coloring problem which satisfies all hard constraints and minimizes the paint cost function. Valued-CSPs are a form of Constraint Optimization Problem (COP) [Dec03].

There are many other extensions of Constraint Satisfaction Problems with other problem solving goals. Some examples of these extensions are modeling counting solvers, Maximum Satisfiability (MAX-Sat), and Weighted CSP (WCSP). Model counting solvers determine the number elements in the set of solutions that satisfy all constraints. Max-SAT problems attempt to find the solution that violates the fewest number of constraints. Max-SAT is useful if solutions that satisfy all constraints do not exist. Weighted CSP (WCSP) is similar to Max-SAT, but includes penalties for each constraint and attempts to find the solution that has the lowest penalty. WCSP is used to find the solutions that violate only the least important constraints [dLMS03, Dec03].

Of the decision support methodologies listed so far, CSP is the method that is best aligned to the four aspects of an architectural DSS. As a decision problem representation, graphical constraint satisfaction problems provide a method to represent decision variables, alternatives, and logical constraints between the decision variables. Valued-CSP provides a way to augment a CSP problem with an objective function which satisfies certain properties such as identity and monotonicity[SFV95]. The decision problem representation in CSP is fundamentally non-sequential, since a specific sequence of decision variables is not pre-specified. The representational features of CSP are strongly aligned with our needs for architectural decision support tool.

Structural reasoning in CSP is possible through heuristic sorting algorithms which determines efficient decision evaluation orders for the simulation. Alternatively, some CSP solution approaches use a Tree decomposition algorithm to rearrange the graphical structure of the CSP in order to increase computational efficiency (For example, see the AND/OR approach for solving CSPs: [Dec04, KDLD05, Mat07]).

Simulation of a CSP can be used to attempt to find a single, feasible set of assignments for a decision problem, if it exists. Valued-CSP's can be used to find the single feasible set of assignments for a decision problem which has the highest possible valuation function. With some modifications to existing simulation algorithms, it is possible to return more than one set of feasible assignments for a CSP [RN02].

CSP problems provide some level of decision problem viewing. In some cases, the structural reasoning results can be presented in an AND/OR tree. An AND/OR tree can be used to identify independent subproblems within a CSP's structure. Usually, this information is used in order to reduce computation resource requirements.[BHW81, DM07, Mat07], but Reference [SKC07] claims it could also be used by a decision-maker to gain insight into the independent branches of a decision problem. The survey of CSP literature did not include decision-data viewing methods of tools that present properties of feasible solutions of the CSP problem. This specific need is one of the topics addressed in the Approach in Chapter 3.

2.3.4 Meta-Language-Based Decision Support

Object-Process Network (OPN)

Object-Process Network (OPN) is a meta-language for encoding, enumerating, and evaluating specific architectural design spaces that was introduced by B. Koo in

Reference [Koo05]. OPN is an algebraic systems representation and reasoning language [KSC07b, KSC07a]. In practice, Koo's meta-language has been shown to be a very powerful tool for automating certain architectural model construction tasks [SKC05a, SKC05b, CCC06, Cam07, Roz07, HKZ07, SPL07].

By automating tasks in architectural reasoning and model construction, OPN can be considered an architectural decision support system [SKC07]. OPN uses a bipartite graph of Objects and Processes, connected by relationships to represent the space of alternative constructions of systems. A very simple OPN diagram is shown in Figure 2-9. A more complex example from Reference [SKC05a] is shown in Figure 2-10.

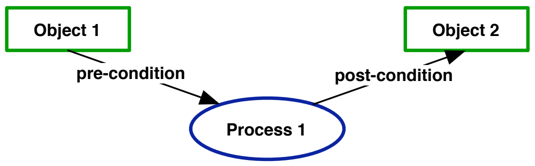


Figure 2-9: A simple OPN. Objects are indicated by green rectangles and processes are indicated by blue ellipses. The relationships between the objects and processes, the pre- and post- conditions, are indicated by arrows.

As a simulation language, OPN satisfies the decision support needs of generating alternative feasible compositions of decisions and evaluating them. The evaluation of models in OPN can be driven by either user-provided metrics, symbolic calculation of performance, or by construction of customized equations on the fly for specific architectural alternatives[1].

Although OPN is a relatively new decision support system it has been applied successfully to several decision problems. In studies supporting decisions for NASA's human Moon and Mars exploration program, it was used to model the mission-mode decision problem [SKC05a], the cargo launch vehicle configuration decision problem [SKC05b] and a NASA stakeholder policy problem [CCC06, Cam07]. In support of study of arctic oil exploration architectures, it was used for an ice protection decision problem, an oil extraction and treating problem, and a process sequencing decision problem [Roz07]. OPN has also been used to study reconfiguration of business processes [XKZ07] and manufacturing processes [HKZ07].

[1]Dynamic metric equation construction was demonstrated in the Apollo study in Reference [Koo05].

As a architectural decision support system, OPN has both strong and weak characteristics. As a representational language, OPN has the ability to *implicitly* encode decision variables, chance variables, metrics, and logical constraints. However, in practice we have found that modeling a decision problem in OPN's Pertri-net-like format connecting objects, processes, and relationships can be conceptually difficult for systems architects. Although OPN implicitly represents the structure of a decision problem, it does not have algorithms for structural reasoning which meet the criteria in Section 2.2.1. The implementation described in References [Koo05], [KSC07c], and [KSC07b] does not explicitly provide the ability to manipulate the structure of the decision problem to increase computational efficiency or to provide information about the connectivity of decision variables. As a simulation tool, OPN can be considered quite powerful, since it provides hybrid numerical and symbolic solvers and can be extended to use external computational tools by importing libraries through its Jython[PR02] script interface. OPN provides some viewing tools that can produce plots of generated feasible architectures, however, there is little guidance as to what type of plots would be useful for supporting architectural decision-making.

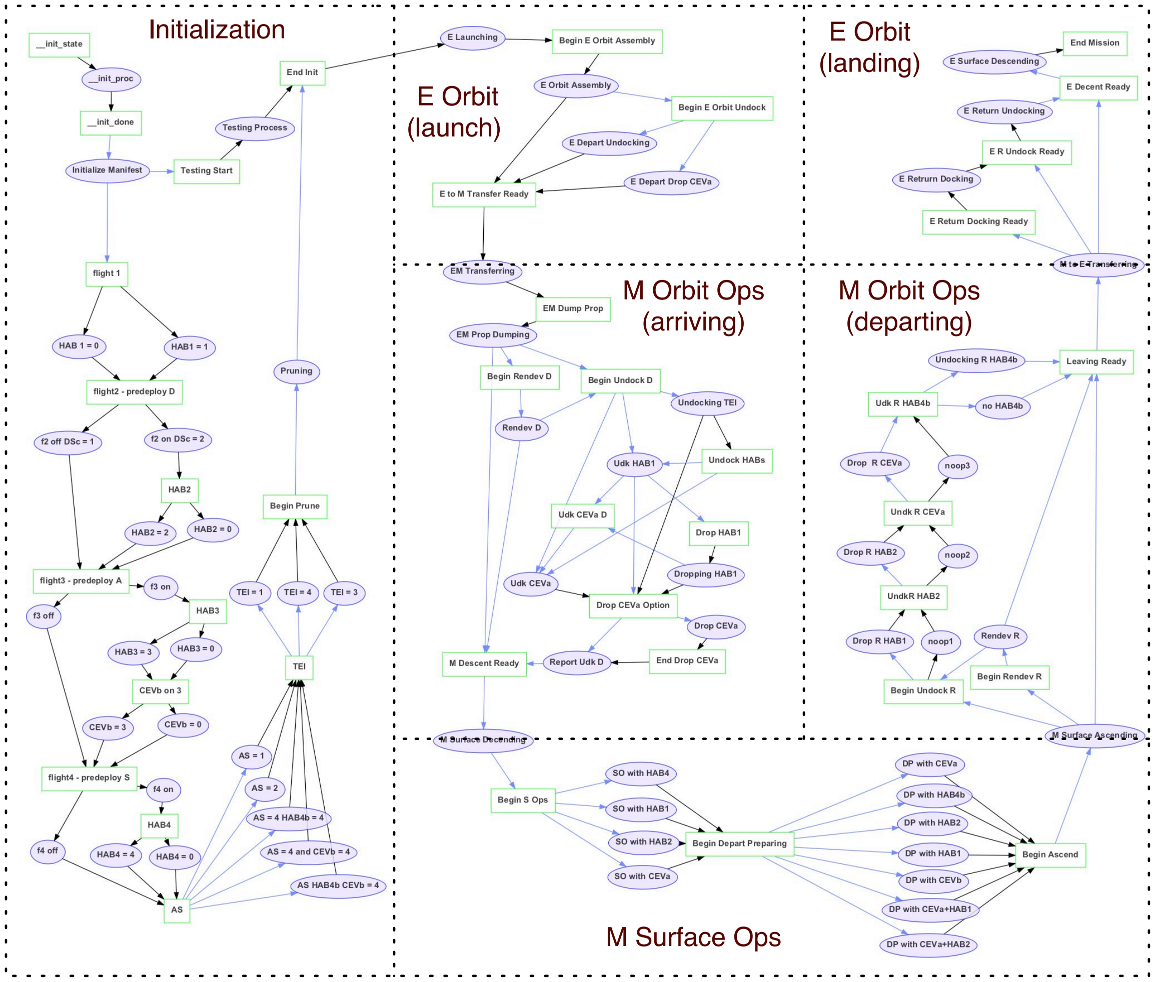


Figure 2-10: An OPN for studying the space of alternative options for mission modes for human moon and Mars exploration. This OPN model was used to partition the infinite space of possible mission mode trajectories into a finite set of 1162 partitions using a motion language [Fra01] abstraction. See Reference [SKC05a] for more details.

2.4 Summary

The goal of this chapter was to identify existing methods from decision support literature and evaluate them against the specific needs of architecture decision-support. The survey of the state of practice (Section 2.3) listed nine different methods for decision support. The representational, structural reasoning, simulation, and viewing aspects of each method were compared against the desirable aspects of a method for architectural decision support listed in Section 2.2.1. Table 2-1 identifies how well each one of the nine methods meets the needs of an architectural decision support system.

Table 2-1: Comparison of nine decision support methods to the aspects of architectural decision support listed in Section 2.2.1. Key: "++" == the aspect of the method meets the needs, "+" = the aspect of the method partially meets the needs.

	Aspects of an Architectural DSS			
	Representing	**Structural Reas.**	**Simulating**	**Viewing**
Morphological Matrix	+			+
Dependency Structure Matrix	+	+		+
Decision Tree	+		+	
Influence Diagram	+	+	+	
Sequential Decision Diagram	+	+	+	
Markov Decision Process	+	+	+	
Time-Expanded Decision Network	+		+	+
Constraint Satisfaction Problems	++	++	+	
Object-Process Network	+		++	+

Two of the methods in Table 2-1 are particularly interesting: constraint satisfaction problems (CSP) and Object-Process Network (OPN). As pointed out in Section 2.3.3, the CSP method has the representational ability to explicitly encode decision variables with alternatives, logical constraints, and metric functions in one graph representation. Additionally, CSP has structural reasoning features which can be used to analyze the structural properties of the decision variables and manipulate the structure of a decision problem to enable more efficient simulation. The simulation aspects of conventional CSP solvers provide for metric calculation of a restricted class of functions, however with some extensions, they could be used to calculate different types of functions. Since CSP was not explicitly invented for architectural decision problems, the literature for CSP lacks specific discussion of methods for viewing the simulation results.

The second interesting method is Object Process Network (OPN). OPN was originally described as a meta-language for systems architecting in Reference [Koo05]. It implicitly represents an architectural decision problem by providing a meta-language with sufficient syntax and semantics for creating a customized domain-specific language to model the space of alternative architectural candidates. Although this approach has been shown to be an effective representation for several architecting problems (see Section 2.3.4), experience with OPN has also shown that it is conceptually difficult for an architect to model architectures using this approach. OPN provides powerful simulation features which can generate a set of feasible solutions and calculate numerical properties through its internal numerical algebraic processor or through external functions calls.

The next chapter presents a framework for supporting architectural decision-making. The features of CSP and OPN are mixed to create a new representation which combines the advantages of the both approaches. In addition, some new architectural decision space viewing tools are developed to provide the ability to present structural reasoning and simulation data to the architect.

3 Architecture Decision Graph

3.1 Overview

In Chapter 1 the motivation for this research and the goals of this thesis were described. Specifically, the needs of the decision-maker were subdivided into four layers: representing, structural reasoning, simulating, and viewing an interrelated decision problem. Chapter 2 connected the four layers of needs with four aspects of desirable features for an architectural decision support system. The features of nine existing decision support methods were compared against these four aspects. Out of the nine methods, two were especially interesting: Constraint Satisfaction Problems and Object-Process Network. In this chapter, the features of these two methods are mixed and supplemented to create decision support framework based on a decision problem representation called Architecture Decision Graph (ADG).

ADG is a decision-focused representation for architecture decision support, built on top of the principles of the morphological method [Zwi66], graphical constraint problem solving [Dec03], and the OPN meta-lanaguage for systems architecting [Koo05]. Formally, an ADG is a bipartite graph representation of an architectural decision problem. ADG provides decision problem structural reasoning algorithms that can extract properties of the decision variables from the structure of the graph. A decision problem represented in ADG is automatically compiled into an Object-Process Network for enumeration of feasible combinations of decisions and for simulation of decision outcomes.

The structure of this chapter is as follows: First, the assumptions and the key contributions from the literature are explained. Next, an overall view of ADG framework is discussed. The subsequent subsections describe the representation, structural reasoning, simulation, and viewing aspects of the ADG framework, with the aid of a simple decision problem involving five interconnected decisions. The chapter concludes with an analysis of ADG's properties, performance and a chapter summary.

3.2 The Architecture Decision Graph Framework

The overall goal of systems architecting is to find satisfactory, or preferable, solutions that are logically feasible and balance competing considerations in a way that is acceptable to the decision maker. Architecture decision graph assists decision-makers by providing the tools to represent a decision space with multiple, interconnected decisions, reason about its structure, and simulate the outcome of possible combinations of decisions.

The specific objective of ADG is to provide a decision support system that satisfies the representation, structural reasoning, simulation, and viewing needs of an architectural decision-maker (Section 1.4). ADG is intended for use during the early system design phase. According to Simon's classification (see Section 1.2), early system design phase problems can be classified as non-programmed decision problems. The ADG method is specifically useful for the "design" activity of decision making: inventing, developing, and analyzing possible courses of action.

3.2.1 Assumptions

The ADG modeling framework is based on the following assumptions:

Assumption 1 *The steps of architecture refinement can be characterized as decisions applied to the architecture space.*

As stated in Section 1.3, the process of systems architecting is a process in which decisions are made to partition and select parts of a design space. Decisions are the steps of architecture refinement which reduce the space of architectural candidates. This view of the design process for engineering systems has been asserted by other authors in references such as [Abr65] and [Cat06]. However, those references did not include a method or tool for developing and analyzing architectural alternatives by representing them as a set of interconnected decisions.

Assumption 2 *Each decision can be represented by a decision variable with a domain that is a finite set of mutually exclusive alternatives.*

The assumption of mutually exclusive sets of alternatives is a standard assumption in decision theory [Han05] and constraint programming [RN02]. This apparent limitation can be overcome by using two different methods. The first method is to construct a set of alternatives for the decision variable that explicitly includes the overlapping combinations as new alternatives. For example, consider a decision variable has the set of choices {A, B, C}. If the decision maker wants to include the choice “A and B” or “A, B and C”, then the set of alternatives can be revised to have these five elements: {A, B, C, AB, ABC}.

The second method is to decompose the decision variable into several decision variables. The decision variable with the domain of assignments {A,B,C} could be replaced by three decision variables: A with the domain {yes,no}, B with the domain {yes,no}, and C with the domain {yes,no}.

Assumption 3 *continuous decision variables can be approximated as discrete decision variables with a finite set of alternatives.*

The goal of ADG is to assist a decision maker in finding categories of satisfactory solutions, while taking into account competing considerations. The goal is not to provide an optimal solution. By first narrowing the decision space to a satisfactory sub-space of the original design space, decision-making and engineering resources can be focused on finding an optimal solution later in the design process.

This assumption limits the precision of ADG’s results, but does not limit its ability to encode decision spaces that include continuous variables. A continuous variable can always be represented by a discrete variable which takes on either certain values or a range of values. Consider a continuous decision variable X, which can take on the values from $-\infty$ to ∞. A discrete approximation for this variable can be transformed into the variable $X_{discrete}$ with the set of just two possible assignments $\{(X < 0), (X \geq 0)\}$. A more specific example is a decision about the budget for a large project. The exact budget may have a range of assignments between 10 and 25 million dollars, however, the range of values can be approximated by the set of specific values: {10 million, 15 million, 20 million, 25 million}.

Assumption 4 *Decisions are not necessarily sequential.*

Many decision support systems assume that there is a fixed order in which decisions can be made. In DSS frameworks like DSM, decision trees, influence diagrams, sequential decision diagrams, and object process networks, decision makers are required to specify a particular order of evaluation for the set of decision variables. For many decision problems, such as problems that deal with planning actions over time, this may be appropriate [RN02].

The ADG framework assumes that decisions can be made in any order. ADG uses this assumption because the restrictions between two decision variables in systems architecting are often bi-directional. From a logical constraint standpoint, it doesn't matter which decision one makes first, because the order will not change the decision's effect on the feasible domain of alternatives of a connected decision.

For example, consider two related decisions:

1. What will I cook for dinner?

2. What ingredients will I buy?

This set of two decisions appears to be temporally constrained: choosing the recipe before going shopping seems to make more sense. In a representation such as a DSM, influence diagram, SDD, or decision tree, the decision-maker would have to choose which decision comes first. However, even though these two decisions are related, there is no reason why these two decisions must be in the order 1 then 2, or 2 then 1. It may make more sense to go to the market and buy what's fresh before choosing which recipe to cook. The feasibility constraint between the two decisions is bi-directional: the recipe for the meal must match the food that is purchased. By writing the constraint in this way the sequence of decisions does not have to be pre-specified in the decision representation.

The intention of this assumption is to simplify the problem specification of architectural decision problems. The ordering can be derived from the structure of the problem rather than by the judgement of the decision-maker.

3.2.2 Key Contributions from Previous Literature

The concept[1] for the Architecture Decision Graph combines several attributes and ideas from the methods and tools listed in Chapter 2. Specifically, ADG builds upon ideas from the Morphological Matrix, constraint networks, and Object Process Networks. The breakdown of the contributions from these methods and tools is listed in Table 3-1.

[1]a mapping of a system's function to its form [Cra07]

Table 3-1: Previous work leveraged for building the ADG framework.

	Aspects of a DSS		
Method/Tool	Representing	Structuring	Simulating
Morphological Method	✓		
Constraint Satisfaction Problems	✓	✓	
Object-Process Network (OPN)	✓		✓

The morphological method [Zwi66] provides the basic ideas for partitioning a system architecture possibility space in terms of a set of decisions with mutually exclusive sets of alternatives. Specifically, the morphological method provides the morphological matrix, an easy-to-understand table specifying the decision variables and the alternative assignments. In ADG, the morphological matrix is used as one of the possible decision problem entry methods.

Constraint Satisfaction Problems offer a way of encoding the interconnectivity of non-sequential decision variables. Constraint graph methods also provide algorithms for structural reasoning over a decision problem [DP89, Mat07, KDLD05].

Object Process Network (OPN) provides a powerful enumeration engine for generating feasible sets of combinations of decisions as well as a simulation engine that provides hybrid symbolic/numeric evaluation of metrics. OPN also provides the basic toolset for representing a system in a bipartite graph. OPN's tools, written in Java, were modified to represent a decision problem in a bipartite graph, rather than OPN's standard directed graph Petri-net-like representation.

3.2.3 Overview of the ADG Reasoning Cycle

The purpose of ADG is provide a representation of an architecting problem in terms of decision variables, logical constraints, and property functions. The purpose of the entire ADG framework, which includes representational, structural reasoning, simulation, and viewing aspects, is to provide a way to explore an architectural candidate space and gain insight into the architecture problem. Exploring the architectural candidate space involves determining the feasible combinations of decision variable assignments and their properties. Gaining insight into the architecture problem involves determining properties of the architectural decision variables, such as their potential impact on system properties or the set of available alternatives for other decisions.

The ADG framework can be described as a cycle that starts with available knowledge about an architectural candidate space, processes it, then produces new available knowledge about the architectural candidate space. The new available knowledge could be used to revise the representation of the architecture problem. This iterative application of the ADG framework is called the ADG cycle.

The ADG cycle is presented in Figure 3-1. The tasks of the cycle are described briefly in this section, then explained in more depth in subsequent sections. Figure 3-1 uses the OPM/OPN convention of objects (green rectangles) to represent steady states and processes (blue ellipses) to represent transformations. The beginning of the cycle starts with available knowledge. Available knowledge includes all information known about the system and its context. Of specific interest to modeling in ADG are decisions that need to be made and the relationships between them.

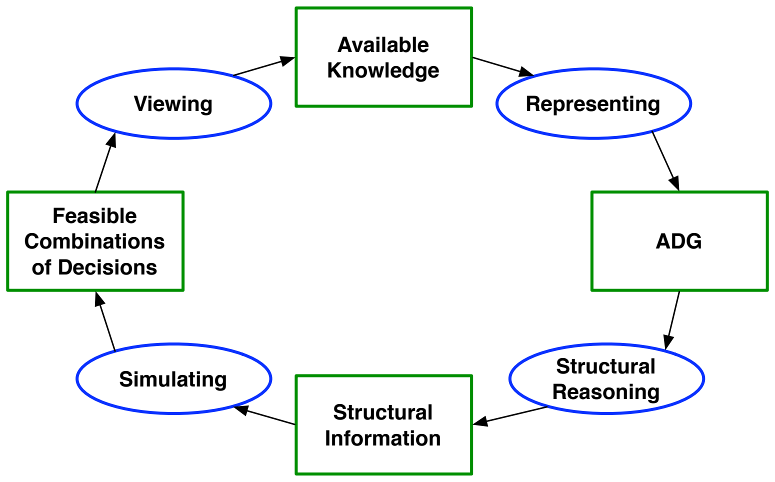


Figure 3-1: The ADG reasoning cycle. Objects are represented by rectangles and processes are represented by ellipses. The cycle begins with available knowledge and ends with increased available knowledge.

The first process in the cycle is **representing**. This process transforms available knowledge about the decision space into an ADG bipartite representation. This is discussed in Section 3.3.

The second process is **structural reasoning**, which transforms an ADG into a structured representation. This is achieved through one of several sorting algorithms described in Section 3.4.

The third process is **simulating**, which generates the set of all feasible combinations of decisions and evaluates them. This process is discussed in Section 3.5.

The last process is **viewing** the decision information. This process is a decision data visualization process. Representing decision-making information using the ADG methodology is a way of generating new available knowledge. This is discussed in Section 3.6.

3.3 Representing

Referring to Figure 3-1, **representing**, is the first process in the ADG cycle which captures and transforms the available knowledge about the decision problem by encoding it in a bipartite graph. In ADG, a decision problem can be represented as a collection of two kinds of things: *system variables*, which include decision variables and property variables, and *relations*, which include the logical constraints and property functions among those system variables. This breakdown of the two types of nodes in an ADG is shown in Figure 3-2. ADG's notation follows the convention of square "objects" and elliptical "processes" from the Object-Process Methodology (OPM)[Dor02] and Object Process Network (OPN)[Koo05]. System variables are represented by objects and relations are represented by processes.

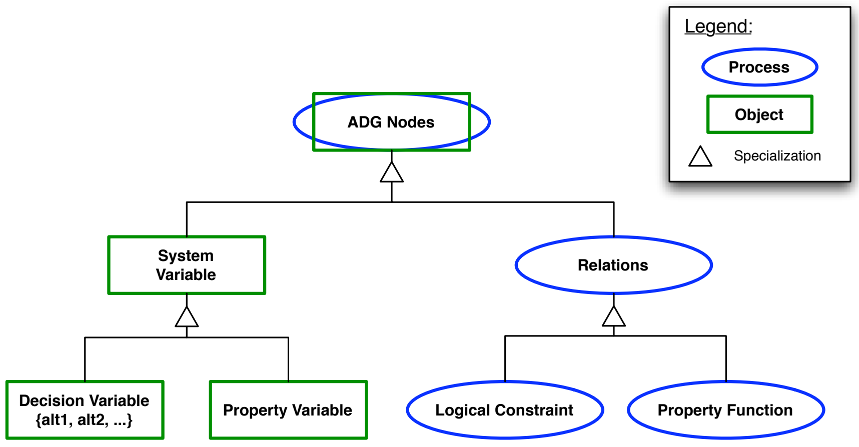


Figure 3-2: An OPD[Dor02] of the two types of ADG Nodes: system variables and relationships.

System variables can be split into two categories: *decision variables*, which are controlled by the decision-maker, and *property variables*, which are derived from the calculation of metrics. The domain of a decision variable is a finite set of alternatives that can be assigned to that variable. Property variables can have either discrete or real-valued continuous domains. An example of a decision variable is "number of wheels for a car", where the domain is the set of alternatives $\{3, 4\}$. An example of a property variable is “cost”, where the domain consists of continuous real numbers greater than zero.

The *relations* between system variables can either be *logical constraints*, which are propositional statements that specify the feasible assignments to two or more decision variables, or *property functions* that specify a metric function to calculate property variables.

The connection between the system variables and relations of the decision problem can be represented in a bipartite graph. Bi-partite graphs have two types of nodes, and connections are only allowed between nodes of different types. In this case, we use the conventions of OPN and call the first type of node “objects”, and represent them with squares, and the second type of node “processes”, and represent them with ellipses. Objects in the ADG represent the two categories of system variables and processes represent the two categories of relationships. This breakdown is shown in Figure 3-2.

3.3.1 Formal Definitions

Below are four definitions: Architecture Decision Graph (ADG), and three types of property functions.

Definition 1 (ADG) *An ADG bipartite graph is defined by the tuple* $ADG = \langle D, M, L, F, E \rangle$ *where,*

- D *is a finite set of* n_D *decision variables,* $D = \{d_1, d_2, ..., d_{n_D}\}$. *Each decision variable* d_i *has a domain consisting of a finite set of* n_{A_i} *mutually exclusive alternatives,* $A_i = \{a_{i,1}, a_{i,2}, ..., a_{i,n_{A_i}}\}$, *associated with it. An* assignment *of decision variable* d_i *is written* $d_i = a_{i,k}$ *where* $a_{i,k} \in A_i$.

- M *is a finite set of* n_M *property variables,* $M = \{m_1, m_2, ..., m_{n_M}\}$. *Property variables are always associated with one specific property function in the set of property functions,* F *(defined below).*

- L *is a finite set of* n_L *logical constraints* $L = \{l_1, l_2, ..., l_{n_L}\}$. *Logical constraints are propositional statements written as a function of the set of alternatives,* A_i *for one more more decision variables.*

- F *is a finite set of* n_F *property functions,* $F = \{f_1, f_2, ..., f_{n_F}\}$. *Property functions are always associated with a specific property variable. Property functions* *relations are either scalar functions or discrete functions of the decision variables in D.*

- *E is a finite set of n_E edges, $E = \{e_1, e_2, ..., e_{n_e}\}$ that connect the object nodes, D and M to the process nodes, L and F. Because ADG is a bipartite graph, object-object edges and process-process edges are not allowed. Edges are a directed edge from a relation to system variable. Edges are defined by a tuple containing a source and a target: $e_k = \langle source \in (L \cup F), target \in (D \cup M) \rangle$.*

Property functions can be defined in three different ways: additively separable, multiplicitively separable, and non-separable. These terms are defined below:

Definition 2 (additive separability) *A property function of n decision variables in the set D is considered additively separable if it can be written as the sum of functions with single arguments. This can be written as:*

$$f(d_1, d_2, ..., d_n) = f_1(d_1) + f_2(d_2) + ... + f_n(d_n) = \sum_{i=1}^{n} f_i(d_i) \tag{3.1}$$

Definition 3 (mutiplicative separability) *A property function of n decision variables in the set D is considered mutiplicatively separable if it can be written as the product of functions with single arguments. This can be written as:*

$$f(d_1, d_2, ..., d_n) = f_1(d_1) f_2(d_2) ... f_n(d_n) = \prod_{i=1}^{n} f_i(d_i) \tag{3.2}$$

Definition 4 (non-separability) *A property function is considered non-separable if it is neither additively separable nor mutiplicatively separable*

The purpose of defining different types of separable functions is because separable functions are easier to implement in the simulation part of the ADG process. Additively and multiplicatively separable functions can be specified in a table. Non-separable functions must be specified in a script. This will be explained further in Section 3.5.

3.3.2 Example Decision Problem

The representing, structural reasoning, and simulating aspects of ADG are best explained using an example decision problem. In this section, an example problem is introduced which has five interconnected decisions, four logical constraints, three property variables, and three property functions. This example problem will be used throughout the chapter to supplement the explanation of the aspects of ADG.

Example: Set of Decision Variables

The five decisions in the example problem are given the generic names decision A, decision B, decision C, decision D, and decision E. A simple way to represent the decisions for the example is to use a morphological matrix, like the one shown in Table 3-2. In this table, each decision is listed with a shortID, a long name, and the units for the set of possible assignments. The shortID is used to identify the decision variable in equations and visualizations. The set of possible alternatives is given in the columns to the right of the decision. For example, Decision A has the possible alternatives of 1, 2, or 3. Decision B has the set of possible alternatives of XS, small, big, or XL.

Table 3-2: Set of five decisions for the example problem presented in a morphological matrix.

shortID	**Decision**	**units**	**alt A**	**alt B**	**alt C**	**alt D**
decA	Decision A	none	1	2	3	
decB	Decision B	none	XS	small	big	XL
decC	Decision C	none	no	yes		
decD	Decision D	none	ROUND	square	hex	
decE	Decision E	none	0	1		

Decision variables could also be entered into the ADG in other ways. One method is to directly create decision variables using the object editor of the OPN-IDE (Object-Process Network-Integrated Design Environment)[Koo05, KSC07c]. ADG's software implementation uses the OPN-IDE software as a graph visualization and modification tool. Another way to enter the set of decision variables is to use a hierarchical morphological matrix. A hierarchical morphological matrix allows decisions which enable or disable other decisions to be represented in a hierarchical structure rather than a flat structure, like the one shown in Table 3-2. This concept is discussed further in Section 6.4.

Example: Set of Logical Constraints

The five decisions in the example problem are connected by a set of relations, which are made up of sets of logical constraints and property functions. The logical constraints in this example are listed in Table 3-4. Logical constraints are encoded using OPN's standard propositional logic syntax. The logical operators in OPN's syntax are defined in Table 3-3.

Table 3-3: Syntax of logical constraints.

symbol	definition	symbol	definition
==	logical equivalence	>	logical greater than
||	logical or	<	logical less than
&&	logical and	>=	logical greater than or equal to
!=	logical not equal	<=	logical less than or equal to

All constraints in Table 3-4 must hold (be equal to "true") for a set of decision variable assignments to be considered feasible. If one or more constraints is violated, then the set of decision variable assignments is considered infeasible. The scope column in Table 3-4 lists the subset of decision variables that are elements included in the equation for each constraint.

Table 3-4: Logical Constraints for the example problem.

shortID	**scope**	**equation**
ABconstraint	decA,decB	(decA == 1) || (decA == 2 && decB == small) || (decA == 3 && decB >= big)
ACconstraint	decA,decC	(decA == 1) || (decA !=1 && decC == yes)
CDconstraint	decC,decD	(decD == hex && decC == no) || (decD !=hex)
BCEconstraint	decB,decC,decE	(decC == no && decB <= small) || (decC == yes && decE == 1 && decB >= big)

One method for constructing the equations for a set of logical constraint is by using constraint tables. Table 3-5 shows two constraint tables for the logical constraints in the example problem. A constraint table indicates the feasible and infeasible combinations of decision assignments for two decisions. An entry of 1 in the table includes a feasible combination of decision assignments and an entry of 0 indicates an infeasible combination.

The approach of using constraint tables for describing the feasible and infeasible variable combinations is similar to the approach used in SpecTRM, The Specification Tools and Requirements Method. References [LHR99] and [Lev00] claim that by using tables to specify all allowed and disallowed combinations of states of the system, the specification process for critical software systems is less error prone. By building a table, a human user can easily detect any combination of variable assignments that have not been explicitly specified as feasible or infeasible, since the entry in the table will be blank. Furthermore, the verification and validation process for the specifications is simplified since a review team can examine and test each entry in the matrix in a systematic way.

Table 3-5: Example logical constraint tables. The constraint equations in Table 3-4 can be derived from these tables. Note that BCEconstraint has two tables since it has a scope of three variables. The BCEconstraint table on the left applies when $decC == no$ and the BCEconstraint table on the right applies when $decC == yes$.

ABconstraint	decA		
decB	**1**	**2**	**3**
XS	1	0	0
small	1	1	0
big	1	0	1
XL	1	0	1

ACconstraint	decA		
decC	**1**	**2**	**3**
no	1	0	0
yes	1	1	1

CDconstraint	decD		
decC	**round**	**square**	**hex**
no	1	1	1
yes	1	1	0

when decC==no:

BCEconstraint	decE	
decB	**0**	**1**
XS	1	1
small	1	1
big	0	0
XL	0	0

when decC==yes:

BCEconstraint	decE	
decB	**0**	**1**
XS	0	0
small	0	0
big	0	1
XL	0	1

Note that the process of translating a constraint table into the OPN syntax for a logical constraint is not automated. The logical equation must be written manually after the the feasible set of combinations of variable assignments is specified by the decision-maker. However, it is feasible to automate this process. Section 6.4 discusses this topic further. Automation of the translation of logical constraint tables into logical constraint equations should reduce the chance of errors when specifying the decision problem.

The table method works well for constraints with a scope of two variables. However, when there are three or more variables involved in the constraint, it is usually easier to write the equation directly. (For example, the BCEconstraint in Table 3-4 has a scope size of three.)

Example: Set of Property Variables and Property Functions

There are three property variables in the example decision problem: cost, risk, and intensity. Each of these has an associated property function: costFunc, riskFunc, and IntensityFunc. For illustrative purposes, these three property functions are constructed using one of each of the three different types of functions. Cost is an additive separable function, risk is a multiplicative separable function, and intensity is a non-separable function.

The two separable functions, costFunc and riskFunc can be described using a "separable property table", like the one shown in Table 3-6. In this case, the table lists the constants that can be added or multiplied to calculate cost and risk. In this case, all of the entries in the table are constants. However, these could also be more complex functions, such as $decA^2$ or $\log(decA)$.

Table 3-6: Example Separable Property Table.

CAT	shortID	Decision	type	alt A	alt B	alt C	alt D
decision	decA	Decision A	none	1	2	3	
prop	cost		add	5	6	7	
prop	risk		mult	0.99	0.95	1	
decision	decB	Decision B	none	XS	small	big	XL
prop	cost		add	1	2	3	5
prop	risk		mult	0.99	0.98	0.95	0.9
decision	decC	Decision C	none	no	yes		
prop	cost		add	2	1		
prop	risk		mult	1	0.99		
decision	decD	Decision D	none	ROUND	square	hex	
prop	cost		add	10	5	30	
prop	risk		mult	0.95	0.9	0.8	
decision	decE	Decision E	none	0	1		
prop	cost		add	1	2		
prop	risk		mult	0.99	0.98		

An example of a cost calculation for decA=2, decB=small, decC=yes, decD = hex, and decE=1 is:

$$costFunc(2, small, yes, hex, 1) = 6 + 2 + 1 + 30 + 2 = 41$$

For the same decision variable assignment, the risk calculation is:

$$riskFunc(2, small, yes, hex, 1) = (0.95)(0.98)(0.99)(0.8)(0.98) = 0.72$$

The third property function, IntensityFunc, is a function that is non-separable. It is defined using the Jython scripting language in Listing 3.1. The Jython[PR02] scripting language is used because it is a standard feature of OPN. The intensity function takes three decision variables for its input: decB, decC, and decD. The output is a floating point number. This type of property function is included here as a demonstration of OPN's ability to evaluate arbitrary user-defined functions. The Jython scripting environment can also be used to call evaluation functions in external libraries or tools, such as Matlab or Mathematica.

Listing 3.1: intensity.py

```
import math as m;

def intensityFunc(decB,decC,decD):
    ## intensity calculation for example problem
    intensity = 0.0;
    x = (decB == small) * 4 + (decB == big) * 7 + \
        (decB == XS) * 2 + (decB == XL) * 2;
    y = (decC == no) * 1.2 + (decC == yes) * .7;
    z = (decD == ROUND) * 1.05 + (decD == square) * \
        1.1 + (decD == hex) * 1.5;

    intensity = m.log(x) + m.log(y)*z;

    return intensity;
```

Visualizing the example problem's ADG

The three visualizations of the ADG representation are shown in Figures 3-3, 3-4, and 3-5. The "complete view" in Figure 3-3 shows all variables and relations in ADG's tuple, $ADG = \langle D, M, L, F, e \rangle$. The second view, Figure 3-4, is the "logical view". It shows only decision variables, and logical constraints (property variables and property functions are excluded). The third view, Figure 3-5, is the "properties view" and it shows only decision variables, property variables, and property functions (logical constraints are excluded). The logical view and property view are useful when building an ADG because it reduces the number of objects and processes that appear in the complete view.

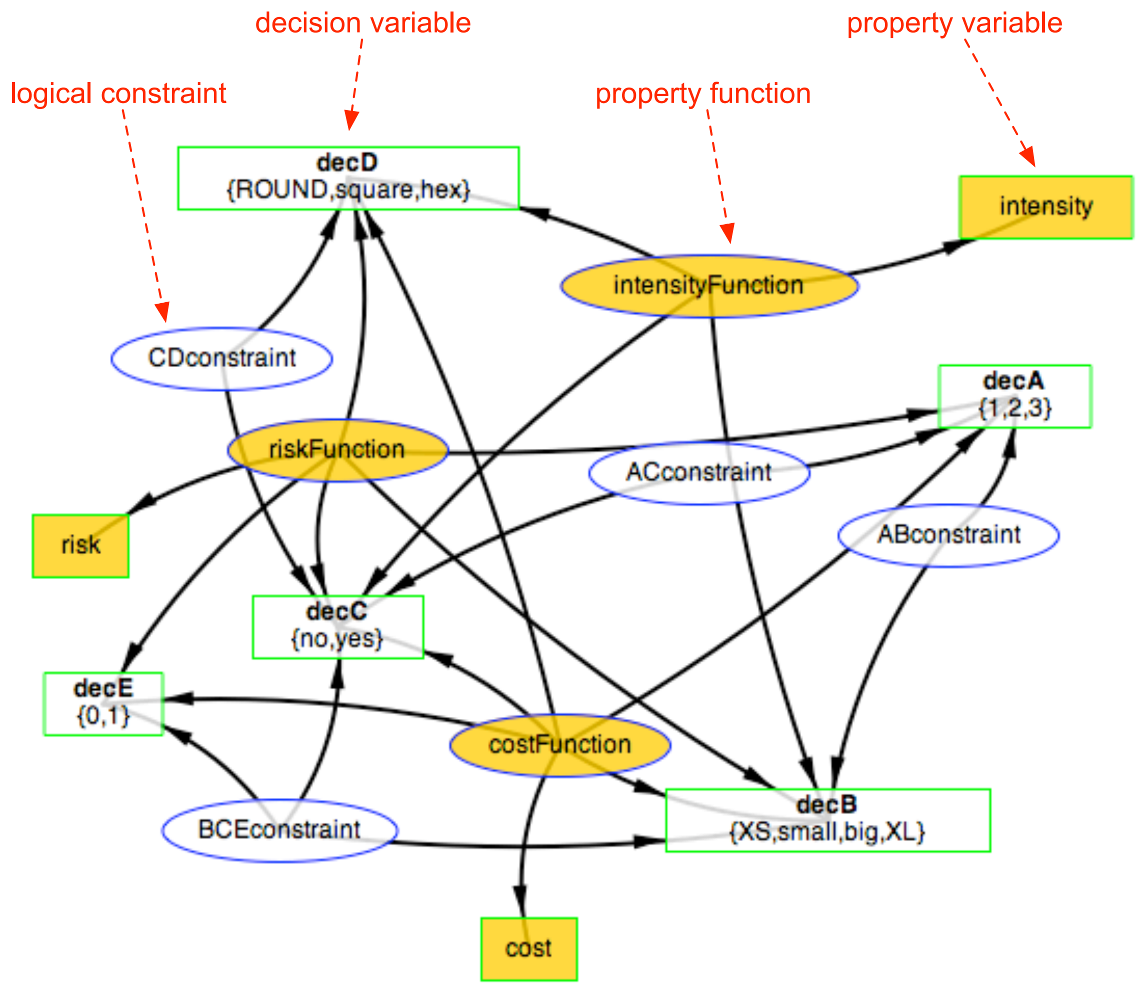


Figure 3-3: ADG example – Complete View. This is a visualization of the ADG bipartite graph. System variables are green rectangles. The decision variables have a clear background and list the set of alternatives in braces below the decision variable's name. The property variables are green rectangles with an orange background. Relations are indicated by blue ellipses. The logical constraints have a clear background, and the property variables have an orange background.

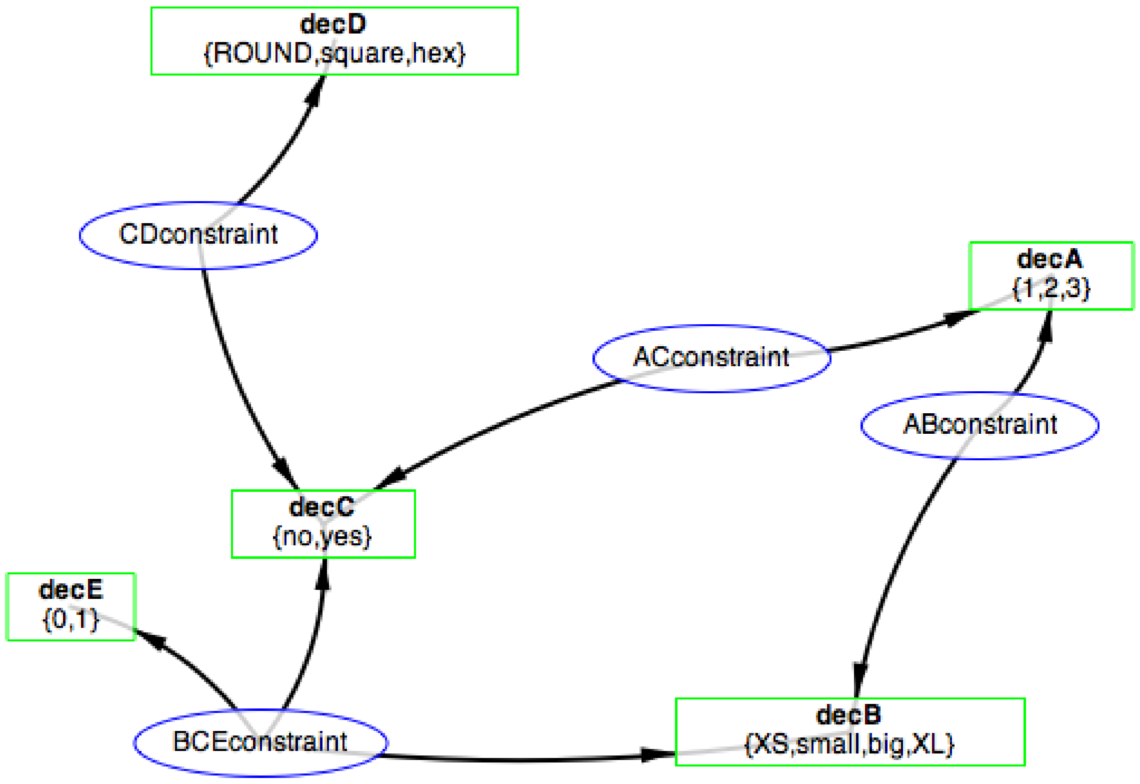


Figure 3-4: ADG example – Logical View. This is an alternative visualization of the ADG bipartite graph. In this case the property variables and property functions have been removed. Only the decision variables and logical constraints appear. This view is useful for visualizing only the logical constraint relations between decisions variables.

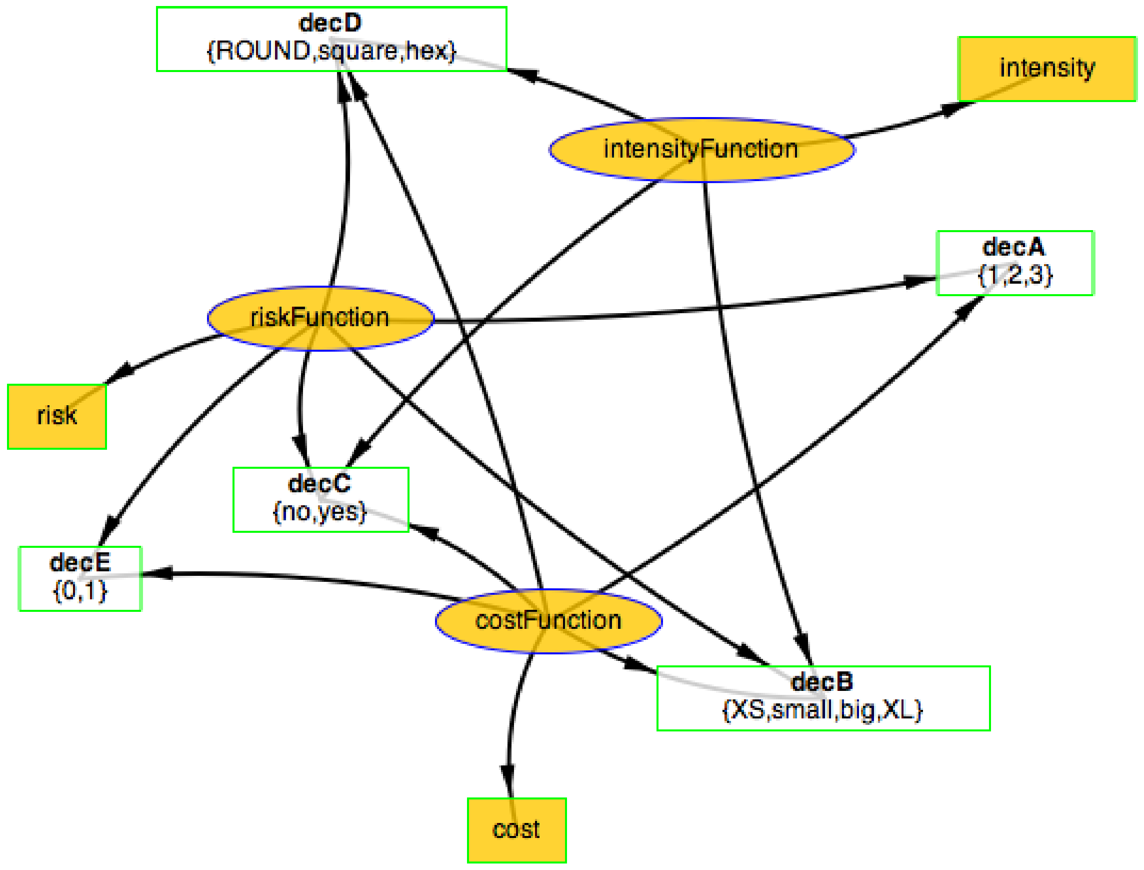


Figure 3-5: ADG example – Properties View. This is another alternative visualization of the ADG bipartite graph. In this case the logical constraints relations have been removed. Three types of nodes remain: property variables, property functions, and decision variables. This view is useful for visualizing only the structure of the property value calculations.

3.3.3 Section Summary

This section formally defined ADG's bipartite formulation for a decision problem. The five types of elements (four types of nodes and one type of edge) that make up an ADG were defined. An example problem with five decision variables, four logical constraints, three property variables, and three property functions was introduced. Examples of additively separable, multiplicitively separable, and non-separable property functions were included in the example decision problem.

The three types of ADG views were presented at the end of the section: the complete view, the logical view, and the properties view. These views can be used as alternate ways to visualize the connectivity of the decision and property variables.

The following section discusses analyzing the structure of the ADG.

3.4 Structural Reasoning

Referring to Figure 3-1, **structural reasoning** is the second process in the ADG cycle. It transforms the information in the ADG bipartite graph into a sorted set by reasoning about its structure. The purpose of structural reasoning is to increase simulation performance and to identify properties of decision variables related to their connectivity to other decision variables. The structural reasoning aspect includes the methods and tools for reasoning about the structure of the decision problem itself. This includes determining which order decision should be analyzed and the degree of connectivity between different decision variables. Structuring is achieved by heuristic sorting methods.

The ADG representation is intentionally an atemporal representation of a decision problem so that the decision ordering can be automated through analysis of the structure of the problem, rather than requiring the order of the decision variables to be pre-specified. By assigning the order as a result of computation, it can be reset if the information in the ADG changes at a later stage in the architecting process. This approach is similar to the graphical constraint problem approach (Section 2.3.3), which is also un-sequenced, and different from the decision tree/influence diagram/sequential decision diagram approach (Section 2.3.2), which is requires a pre-specified variable sequence.

Since ADG doesn't require the decision-maker to assume that he knows which decisions are made before others, the algorithms can determine a computationally efficient decision variable order in which to prepare the simulation. As described in Chapter 1, a key challenge in architectural decision-making is that the many-to-many relationships often lead to massive search spaces which are difficult to handle for both humans and computers. A key concept in reducing the effort needed to explore these search spaces is to reduce the amount of early branching of the search space.

Research in the field of autonomous reasoning has demonstrated that the effort required to search a solution space can be reduced by picking certain variables orders over others[RN02]. Specifically, in the field of constrained graph problem solving (Section 2.3.3), heuristic methods have been identified which reduce branching in the early stages of search. A heuristic is a (usually) simple method or algorithm which approaches a good solution. In idiomatic English, a heuristic method for problem solving is often called a "rule of thumb". In general, heuristics methods provide no guarantee of optimality or completeness. However, in practice, heuristic algorithms are used to find satisfactory solutions with dramatically less effort than provably optimal algorithms. In this thesis, two commonly used variable sorting heuristics from the graphical constraint problem solving literature are adapted to work with ADG's bipartite formulation of decision problems.

The first heuristic that is adapted in this thesis is commonly called the "degree heuristic." The degree heuristic sorts decisions variables in the order of most constrained to least constrained. In graphical problem solving literature [RN02], the degree of connectivity for a decision variable is usually measured by simply counting the number of constraint equations that are included in the decision problem's logical constraint equations. This heuristic reduces branching in the decision space because the alternatives for highly constrained decision variables are more likely to be eliminated early in the decision problem analysis.

The second heuristic is often called the "minimum remaining values heuristic". This heuristic ranks decision variables in the order of fewest number of alternatives to greatest number of alternatives. Since each alternative potentially creates a new branch in the search space, it's advantageous to delay the analysis of decision variables with many alternatives until the end of simulation process.

There are two sorting algorithms for structural reasoning explained below: ADGsort1 and ADGsort2. Each one analyses the structure of the decision problem in order to provide a computationally efficient sequence for the simulation aspect.

3.4.1 ADGsort1

ADGSort1 (Algorithm 3.1) is the first sorting algorithm. It sorts the decision variarbles by their *degree of connectivity.* This is modification of the "degree heuristic" [RN02]. The input is to ADGsort1 an ADG and the result is an ordered list of decision variables. Instead of using the degree of the node (defined as the number of connected edges), the degree of connectivity is measured by counting the number of decision variables connected through adjacent logical constraints.

A logical constraint is considered *adjacent* if it is connected via an edge to the decision variable. A decision variable is "connected through an adjacent logical constraint" if it's directly connected through an edge to the set of all adjacent logical constraints. In Figure 3-4, the decision variable decE is adjacent to the logical constraint BCEconstraint. The decision variable decE is connected to the decision variables decC and decB through adjacent logical constraints. Note that property variables are not counted in this calculation because they do not effect branching in the decision space. The algorithm for determining the degree of connectivity is given below:

Algorithm 3.1: ADGsort1

```
   Data: $ADG = \langle D, M, L, F, E \rangle$
   Result: $degC$, a map relating each decision in $D$ with its degree of
           connectivity.
 1 forall decision variables $d_i \in D$ do
 2     Initialize Set of connected decisions, $D_{in,i} = \{\}$;
 3     Initialize degree of connected decisions $degC_i = 0$;
 4     forall logical constraints $l_j \in L$ connected to $d_i$ do
 5         forall $d_k$ connected to $l_j$ do
 6             $D_{in,i} + d_k$;
 7         end
 8     end
 9     $degC_i = size(D_{in,i}) - 1$ ;
10 end
11 return $degC$;
```

In the small example problem, the degree of connectivity can be calculated by hand by referring to the logical view of the ADG in Figure 3-4. The degree of connectivity of decA is 2 since it is connected to decC through ACconstraint and decB through ABconstraint. The degree of connectivity for decB is 3 because it is connected to decA through ABconstraint, and decC & decE through BCEconstraint. The remaining degrees of connectivity for the example problem are presented in the following table:

Table 3-7: Results of ADGsort1 for the example problem

decision	**degree of connectivity**
decA	2
decB	3
decC	4
decD	1
decE	2

The result of ADGsort1 implies the decision variable decC is the most connected and decision variable decD is the least connected. The suggested order of evaluation according to ADGsort1 is {decC, decB, decA, decE, decD}. Note that, alternatively, the ordering {decC, decB, decE, decA, decD} could be the result of ADGsort1 since decA and decE have the same degree of connectivity. In this thesis, this kind of ambiguity by using the sequence variables were inserted into the set of decision variables, D. In this case decision variables were inserted in alphabetic order.

The performance impact of ADGsort1 on the simulation aspect of ADG is discussed below in Section 3.7.2.

3.4.2 ADGsort2

ADGsort2 (Algorithm 3.2) is an enhancement of ADGsort1. It adjusts the suggested order of the decision evaluation by subtracting the number of alternatives for each decision from the degree of connectivity. This approach is similar to combining the "degree heuristic" and the "minimum remaining values heuristic" into a single heuristic. The intent of ADGsort2 is to further reduce the amount of branching required in the early stages of ADG simulation by pushing variables with many alternatives and low degrees of connectivity to the end of the decision evaluation order. The algorithm for ADGsort2 is given below:

```
Algorithm 3.2: ADGsort2
Data: $ADG = \langle D, M, L, F, E \rangle$
Result: $sort2rank$, a map relating each decision in $D$ with a measure equal to
        a decision variable's degree of connectivity minus its number of
        alternatives.
1 Initialize degree of connectivity, $degC$, using ADGsort1;
2 forall decision variables $d_i \in D$ do
3     $A_i$ is the set of alternatives for $d_i$;
4     $sort2rank_i = degC_i - size(A_i)$ ;
5 end
6 return $sort2rank$;
```

The result of ADGsort2 for the example problem is given in the table below. For reference, columns for degree of connectivity and number of alternatives are also included in the table.

Table 3-8: Results of ADGsort2 for the example problem

decision	degree of connectivity	# of alternatives	**sort2rank**
decA	2	3	-1
decB	3	4	-1
decC	4	2	2
decD	1	3	-2
decE	2	2	0

In this case, the suggested order of evaluation according to ADGsort2 is {decC, decE, decA, decB, decD}. Again, the ambiguity between decA and decB is resolved using alphabetic order. The performance impact from using ADGsort2 is also discussed in Section 3.7.2.

3.4.3 Section Summary

This section described two sorting algorithms, ADGsort1 and ADGsort2, that provide structural reasoning for a decision problem encoded in an architecture decision graph. Structuring is used to determine an efficient ordering of decision evaluation.

Note that since ADG is a un-sequenced representation of a decision problem, ordering the decision variables using one of these two algorithms is not required.

It is possible to specify any arbitrary order and still satisfy the input needs of the simulation algorithm. In subsequent sections, test data will demonstrate that that simulation effort can be reduced by using one of these two sorting algorithms.

The next section describes the simulation aspects of the ADG methodology. In Section 3.7.2, the impact of the two sorting algorithms is discussed.

3.5 Simulating

Referring to Figure 3-1, **simulating** is the third process in the ADG cycle which transforms the structured decision problem into an executable OPN model. In this part of the implementation, an executable OPN model is created from the system variables and relations encoded in the ADG representation. Then, the OPN model is executed to produce a set of feasible combinations of decisions and their calculated properties.

This section is divided into two parts. The first part is OPN Compilation, which describes the algorithms for the construction of an OPN model based on the ADG bipartite graph. The second subsection is the description of model execution, which produces the set of feasible combinations of decisions.

3.5.1 OPN Compilation

The structural reasoning process described in Section 3.4 can be thought of as a pre-compilation method for the decision problem before it is compiled into an executable OPN model. By structural reasoning the ADG before compilation, the construction of the model can be done such that the execution of the OPN requires less computational effort. Since the sorting methods ADGsort1 and ADGsort2 are heuristic methods, they are not *guaranteed* to produce an efficient model. However, test results show that they *tend* to produce more efficient models in terms of memory consumption and computational time (see Section 3.7.2).

The construction of the OPN model from the ADG is an automated compilation process. It is achieved by first constructing "blocks" of OPN models for each decision variable. Examples of OPN blocks for decC and decE are shown in Figure 3-6. After each of the blocks are connected, logical constraints and property functions are added. When complete, this executable OPN model is used to enumerate and evaluate the set of feasible combinations of decisions. Pseudocode for OPNBuild1 is presented in

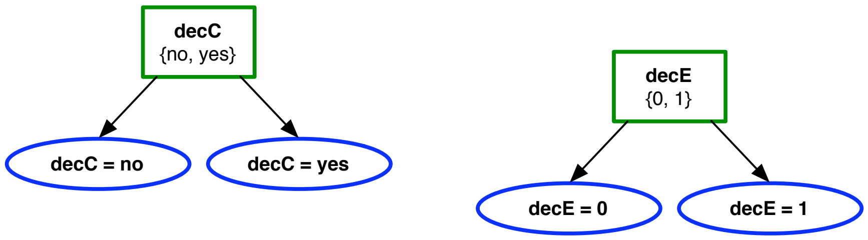


Figure 3-6: OPN building blocks for decC and decE. Decision variable decC can be assigned decC=no or decC=yes. Decision variable DecE can be assigned decE=0 or decE=1. These building blocks are assembled for each decision by the OPNBuild1 algorithm. These two building blocks are highlighted by red-dashed boxes in Figure 3-7.

Algorithm 3.3.

An overview of the OPNBuild1 algorithm for building the OPN executable model from an ADG is outlined below. The OPN model which is produced by the algorithm for the example problem is shown in Figure 3-7.

Refering to the psuedocode on page 73, the first part of the algorithm is from lines 1 to 9. In the first operation, a blank OPN model is created. Second, an initialize object, an initialize process, and final object are added to the OPN. Next, building blocks for each decision variable are inserted into the new OPN in the sequence of either ADGsort1, ADGsort2 or some other arbitrary decision variable ordering. The last building block is connected to an object named "configuration complete". "Configuration complete" is then connected to a process called "metricCalc". Then, the property functions for each non-separable property are added to a process named "metric calculation". At this point all necessary objects, processes, pre-conditions and post-conditions have been added to the OPN model.

In the next section of the pseudocode, lines 10 to 12, all logical constraints are inserted into the OPN post-conditions at specific locations in the model. The locations are chosen such that logical constraints are inserted into the earliest post-conditions after every decision variable in the constraint's scope has been assigned.

In the last section, lines 13-22, property functions are added to the model. Non-Separable properties are inserted into the "metricCalc" process. Separable property calculations are added to each process in which a decision variable is set.

The result of OPNbuild1 is returned on line 22. It is an executable OPN model which can be used to enumerate and to evaluate all feasible combinations of decision assignments. For the example decision problem, the final OPN is shown in Figure 3-7.

Algorithm 3.3: OPNbuild1

```
   Data: $ADG = \langle D, M, L, F, E\rangle$, and an ordering of decision variables in the
         set $D$. It can be either $degC$ from ADGsort1, $sort2rank$ from
         ADGsort2, or another total ordering of the decision variables
   Result: $OPN_{ds}$, an executable OPN model of the decision space.
   /* add all objects, process, pre-conditions and post-conditions
      */
 1 Initialize a new OPN model, $OPN_{ds} = \langle P, B, \langle obj, proc, pre, post\rangle\rangle$ ;
 2 Add the initialize object __I , initialize process initialize , and final object
   __F to $OPN_{ds}$;
 3 forall decision variables, $d_i \in D$ do
 4     Insert a building block for each decision in the order derived from the
       pre-compilation. (An example of a building block for decision $D_1$ is shown
       in figure 3-6.) The connections to the next decision, $D_2$ are shown here as
       well. ;
 5 end
 6 Connect the last building block to an object named "configuration complete";
 7 Create a process named metricCalc ;
 8 Connect a pre-condition between configuration complete and metricCalc. ;
 9 Connect metricCalc to the end object, __F;
   /* insert all logical constraints into post-conditions           */
10 forall logical constraints, $l_j \in L$ do
11     For each logical constraint, insert copies into the earliest post-condition
       after which all variables in its scope have been completely set. ;
12 end
   /* insert all property functions into processes                  */
13 forall property functions, $f_k \in F$ do
       /* note: property variable $m_k$ is associated with property
          function $f_k$                                             */
14     if type of $f_j$ is non-separable then
15         define the scope of $f_j$ as set $D_{scope} \subset D$ ;
16         insert the property function definition, $f_j(D_{scope})$, into the global script;
17         insert the function call $m_j = f_j(D_{scope}))$, into the metricCalc process;
18     end
19     else if type of $f_j$ is separable then
20         If the property function is separable, insert the increment or multiplier
           in each decision assignment process, as appropriate. ;
21     end
22 end
23 return $OPN_{ds}$;
```

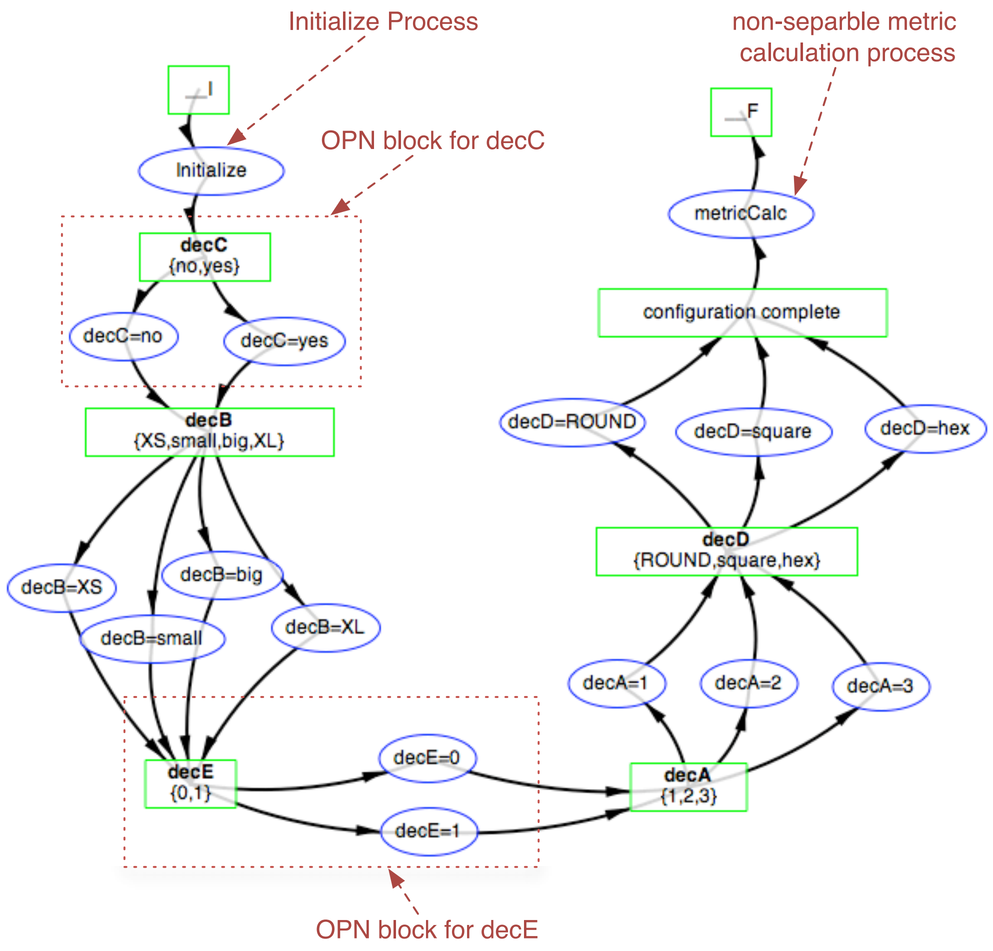


Figure 3-7: OPN Model for the example problem generated using the OPNBuild1 algorithm guided by the sorting from the ADGsort1 algorithm. The OPN Blocks for decC and decE from Figure 3-6 are highlighted with dashed lines.

3.5.2 Model Execution

The model can be executed using the OPN-IDE (OPN-Integrated Design Environment) software or by making a direct function call to the OPN kernel library. The goal of model execution is a list of all consistent decision assignments and their properties.

This result of model execution, $\mathcal{C}$, is called the *set of pairs of feasible combinations of decision variable assignments and their properties.* For brevity, this set is informally called **"the set of feasible combinations of decisions"** throughout this document. Formally defined, $\mathcal{C}$ is a set of pairs, $\{\langle\tilde{D},\tilde{M}\rangle\}$. The first element of the pair is the *set of feasible decision variable assignments*, $\tilde{D}$. It is defined as:

$$\tilde{D} = \{\underline{\tilde{d}} = \langle \tilde{d}_1, \tilde{d}_2, ..., \tilde{d}_{n_D} \rangle \mid \bigwedge_{r=1,...,n_L} l_r(\underline{\tilde{d}}) = \text{true}\}, \tag{3.3}$$

where, $\underline{\tilde{d}}$ is a vector of decision variable assignments (defined in Section 3.3.1) that satisfy all n_L logical constraints in the ADG's set of logical constraints, L. The symbol $\wedge$ is the logical and operator.

The second element in the pair is the *set of associated properties*, $\tilde{M}$. $\tilde{M}$ is defined by:

$$\tilde{M} = F(\underline{\tilde{d}}) \mid (\underline{\tilde{d}} \in \tilde{D}), \tag{3.4}$$

where $\tilde{M}$ is a set of values for all property variables in the ADG's set of property variables, M. Each property is calculated using the set of property functions, F, by applying each function in the set to each element in the set of feasible combinations of decision variable assignments, $\tilde{D}$.

An example of the set $\mathcal{C}$ is presented in Table 3-9. It contains the complete data set of the 20 feasible combinations of decision variable assignments which satisfy all the constraints in the example problem. These twenty combinations are a subset of the entire combinatorial space of the five decisions, which has 144 combinations ($2 \times 4 \times 2 \times 3 \times 3 = 144$). The remaining 124 combinations of assignments violated at least one of the constraints in the set of logical constraints, L.

Note particularly that the decision assignment, decA=2 does not appear in Table 3-9. This is because the assignment decA=2 is never a feasible assignment, given the logical constraints. Referring to the logical constraints table (Table 3-4), this can be traced to the interaction between constraints ABconstraint, ACconstraint, and BCEconstraint. ABconstraint requires that if decA equals 2, then decB equals "small". ACconstraint requires that if decA is not 1, then decC must be equal to "yes". However BCEconstraint does not allow the combination of decB is "small" and decC is "yes". Therefore, decA=2 is eliminated as a possibility.

Table 3-9: The set of feasible combinations of decision variable assignments and their associated properties (cost, intensity, and risk) for the example problem, $\mathcal{C} = \{\langle \tilde{D}, \tilde{M} \rangle\}$. Note that one possibility, decA=2 does not appear in this table since is not consistent with any combinations of the other four decisions. This means that all combinations of decisions that include the decision assignment of decA=2 are infeasible.

decision variable assignments ($\tilde{D}$)					**properties** ($\tilde{M}$)		
decA	decB	decC	decD	decE	cost	risk	intensity
1	small	no	hex	1	41.000	0.761	1.660
1	small	no	ROUND	1	21.000	0.903	1.578
1	XS	no	ROUND	1	20.000	0.912	0.885
1	small	no	square	1	16.000	0.856	1.587
1	XS	no	hex	1	40.000	0.768	0.967
1	small	no	square	0	15.000	0.864	1.587
1	XS	no	square	1	15.000	0.864	0.894
1	small	no	hex	0	40.000	0.768	1.660
1	small	no	ROUND	0	20.000	0.912	1.578
1	XS	no	hex	0	39.000	0.776	0.967
1	XL	yes	ROUND	1	23.000	0.821	0.319
1	XS	no	square	0	14.000	0.873	0.894
3	XL	yes	ROUND	1	25.000	0.830	0.319
1	XL	yes	square	1	18.000	0.778	0.301
3	XL	yes	square	1	20.000	0.786	0.301
3	big	yes	square	1	18.000	0.830	1.554
3	big	yes	ROUND	1	23.000	0.876	1.571
1	XS	no	ROUND	0	19.000	0.922	0.885
1	big	yes	ROUND	1	21.000	0.867	1.571
1	big	yes	square	1	16.000	0.821	1.554

Overview of OPN's Execution Semantics

Precise details of OPN's syntax and semantics as well as the specification for the OPN-IDE software are available in References [Koo05, KSC07c, KSC07b, KSC07a]. For reference in this thesis, a brief summary of the execution semantics of OPN are given below.

The execution semantics of OPN are derived from the concepts in the Petri-net literature [Pet62, Pet81]. Data about the set of feasible designs are carried through the model in a data structure called a *token*. Figure 3-8 shows an annotated OPN diagram showing two objects and one process. In this figure, the tokens are indicated by small circles. Data stored in the tokens are transformed by the process.

A token represents a communication or computation event. When the execution of the OPN model begins the first token is "fired" from the object labeled `__I`. The data in the token is applied to all feasible paths leaving the object. An object is a data storage abstraction. An object may store zero or more tokens. The directed arcs leaving an object are called pre-conditions. A pre-condition may contain a rule which can be evaluated to either true or false, depending on the properties stored in the token. If and only if the pre-condition is evaluated to true, the token is allowed to proceed to the next process.

Processes in an OPN transform the data stored in a token. In Figure 3-8, there are two tokens stored in the object `__I`. One token has the data $\{x = 1\}$ and the other has the data $\{x = 4\}$. The two token are transformed by the process which has the instructions $\{x = x + 1; y = 2\}$. This transformation results in new tokens with the new data $\{x = 2, y = 2\}$ and $\{x = 5, y = 2\}$. However, each new token must pass a post-condition before it is stored in the next object.

A post-condition is a directed arc that connects a process to an object. In Figure 3-8, the post-condition has the rule: $x < 3$. In this case the token with the data $\{x = 5, y = 2\}$ will be eliminated since $x < 3$ is false. Post-conditions are semantically the same as pre-conditions; a rule can be stored in a post-condition that must be evaluated to true for the token to be stored in the next object.

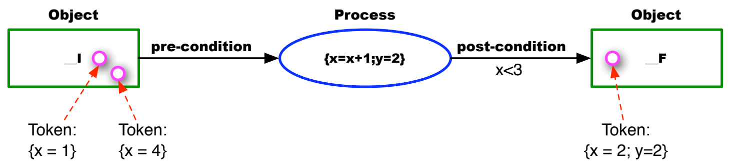


Figure 3-8: An annotated diagram of a simple OPN. Tokens are depicted as circles in this diagram for illustrative purposes. Objects are data storage abstractions and Processes are data transformation abstractions. Pre- and Post-conditions contain propositional rules which must evaluate to 'true' for a token to proceed through the network

3.5.3 Section Summary

This section described how simulation of a decision problem represented in ADG is achieved. The simulation process is composed of two sub-processes: OPN compiling and model execution.

OPN compiling begins after running the structural reasoning algorithms (either ADGsort1 or ADGsort2). Next, the OPNbuild1 algorithm constructs "OPN blocks" containing objects that represent decisions variables and processes that represent the assignment of each of the decision variable's alternatives. These are inserted into an executable OPN model of the decision space, called OPN_{ds}. Next, property functions and logical constraints are added to the model automatically. The OPN model is built in such a way that it is guaranteed to terminate, since there are no cycles allowed in the model. After OPN compilation, this section explained how the OPN model is executed to to produce the set of feasible combinations of decisions. Properties are calculated for each of the feasible combinations.

The following section presents two methods for representing (viewing) the data output by the simulation.

3.6 Viewing

Referring to Figure 3-1, **viewing** is the fourth process in the ADG cycle which transforms the feasible combination of decisions into new available knowledge. The product of simulating a decision problem using ADG is a list of possible combinations of decisions and their calculated properties. For the example problem the list (Table 3-9) contains 20 possible combinations of assignments for the five decisions as well as their cost, risk, and intensity properties. In order for this information to be useful in the decision-making process, it must be presented in a way that is meaningful to the architect. The overall goal of viewing the simulation data is to improve the architect's ability to comprehend the space of feasible decisions.

This section discusses two of the possible views of the decision data. The first is a "decision space view", which plots the decisions in the two-dimensional space measuring their connectivity versus sensitivity to properties. The second view is a "Pareto front view", which is a plot indicating the sub-set of feasible solutions which are non-dominated in terms of pairs of metrics. Other views of the data are possible; some of these are discussed in Section 6.4.

3.6.1 Decision Space View

The decision space view is a plot measuring the impact of decision variable on the decision space. In practice, the words "high-impact decision" could have two different meanings: 1) the decision strongly influences the properties of the system (such as cost, performance, or risk) or, 2) the decision strongly affects the feasible set of alternatives for other decisions. Using ADG, a decision's impact on the system can measured by two different metrics, which correspond to these two definitions of high-impact: *property variable sensitivity* and *degree of connectivity.*

A measure of the property variable sensitivity can be calculated using a modified version of "main effects" analysis that is used in the design of experiments (DOE) literature [BHH05]. The *main effect* is a measure of the average change in system-wide properties produced by changing one variable in a decision problem. It is used to determine which decision variables have the strongest effect on the metrics of the system in terms of delta from a reference configuration.

The traditional formulation of main effects is limited in several ways. Main effects analysis assumes there is a baseline design and the decision variables are modified one at a time. The modification of a decision variable is limited to just two "levels" of assignments and can only be changed one variable at a time. Also, main effects analysis assumes that the system response to the change in a variable is linear. In other words, it is assumed that all variable can be independently modified without changing the feasible space of system configurations.

Since ADG allows more than two "levels" of assignments to decision variables, and explicitly models variable interaction, a modification to traditional main effects analysis is necessary to measure the *property variable sensitivity* of decisions. The property variable sensitivity (PVS) is calculated for each property and each decision over the set of feasible combinations of decisions and their associated properties, $\mathcal{C} = \{\langle \tilde{D}, \tilde{M} \rangle\}$. The equation for PVS is:

$$PVS_{m_j,d_k} = \frac{\sum_{a_{k,i} \in \tilde{A}_k} |E(m_j) - E(m_j | d_k = a_{k,i})|}{|\tilde{A}_k|} \tag{3.5}$$

where:

- PVS is the property variable sensitivity. PVS_{m_j,d_k} is read as the "property variable sensitivity of property m_j to changes in the assignment for decision variable d_k."
- m_j is of one of the n_M property variables.
- d_k is one of the n_D decision variables.
- $\tilde{A}_k \subseteq A_k$ is the set of decision variable assignments for d_k that exist as assignments to decision variables in the feasible set of decision variable assignments $\tilde{D}$. For example: for decision variable decA, $A_{decA} = \{1, 2, 3\}$, but $\tilde{A}_{decA} = \{1, 3\}$,

since the assignment decA=2 does not exist in the feasible set of decision variable assignments.

- $a_{k,i}$ is one of the n_{A_k} alternatives for decision variable d_k. The expression $a_{k,i} \in \tilde{A}_k$ indicates that $a_{k,i}$ is a member of the set $\tilde{A}_k$.
- $E(m_j)$ is the mean of property m_j over all feasible combinations of decisions in $\mathcal{C}$.
- $E(m_j|d_k = a_{k,i})$ is the mean all feasible combinations restricted to include the only combinations with decision assignment $d_k = a_{k,i}$.
- $|\tilde{A}_k|$ is the cardinality (number of elements) of the set $\tilde{A}_k$.

In simpler terms, PVS is a measure of the average magnitude of change in a property that occurs when changing the assignment of particular decision variable.

As an example, the property variable sensitivity of decision A on the intensity property, $PVS_{intensity,decA}$, can be calculated using the the raw data in Table 3-9. First, the overall mean for intensity is: $E(intensity) = 1.131$. Next, the mean when A=1 and A=3 are calculated: $E(intensity|decA = 1) = 0.936$, and $E(intensity|decA = 3) = 1.180$. These numbers are applied to Equation 3.5:

$$PVS_{intensity,decA} = \frac{|1.131 - 0.936| + |1.131 - 1.180|}{2} = 0.122$$

Note that there are three possible assignments for decA: $A_{decA} = \{1, 2, 3\}$. However, since decA=2 was not contained in any element in the feasible set of decisions, $\mathcal{C}$, it was included in the set $\tilde{A}_{decA}$, and therefore was not included in the calculation of $PVS_{intensity,decA}$.

The numerical value of the PVS for intensity to decision A is only meaningful when compared to the PVS for the other decisions in Table 3-10. For example, the sensitivity to intensity of decision A is greater than decision D or E and less than decision B and decision C. This means that a change in the assignment of decision B will potentially vary the intensity property over a wider range than decision A, on average. Qualitatively, by this measure, decision B has a potentially higher impact on intensity than decision A.

One of the key advantages of calculating the property variable sensitivity using Equation 3.5 is that it is calculated over the feasible decision space and variable

Table 3-10: Decision view report for the ADG example problem. The decision view report also includes eliminated alternatives.

DECISION VIEW REPORT:			Sensitivity		
longName	**shortName**	**degConnectivity**	**cost**	**risk**	**intensity**
Decision A	decA	2	1.0625	0.0058	0.1220
Decision B	decB	3	2.2500	0.0157	0.4865
Decision C	decC	4	2.2500	0.0112	0.1627
Decision D	decD	1	8.4000	0.0391	0.0909
Decision E	decE	2	0.9286	0.0095	0.0930

ELIMINATED OPTIONS:
decA=2 appears 0 times.

interactions are taken into account. In traditional sensitivity or mean effects analysis, it is assumed that the model is linear and all decision variable assignments are taken into the calculation without regard to whether or not they are feasible.

The second metric of the potential impact of a decision variable is the *degree of connectivity*. This is easily calculated using ADGsort1 (Algorithm 3.1). It is a measure of how many other decision variables are connected to a particular decision though constraints. This can be considered a first order measure of the impact of one variable on another. A change in the assignment of a decision variable with a high degree of connectivity will potentially affect a larger set of alternatives of connected decisions, in comparison to a decision with a lower degree of connectivity. A decision with a degree of connectivity of zero cannot affect other decisions.

The two metrics, degree of connectivity and property variable sensitivity can be plotted on orthogonal axes in the *decision space view*. The decision space view is a way to visualize both measures of decision impact in one diagram. On the vertical axis is a measure of property sensitivity of each decision. On the horizontal axis is the measure of the degree of connectivity. Examples of decision space view are in Figures 3-9, 3-10, and 3-11.

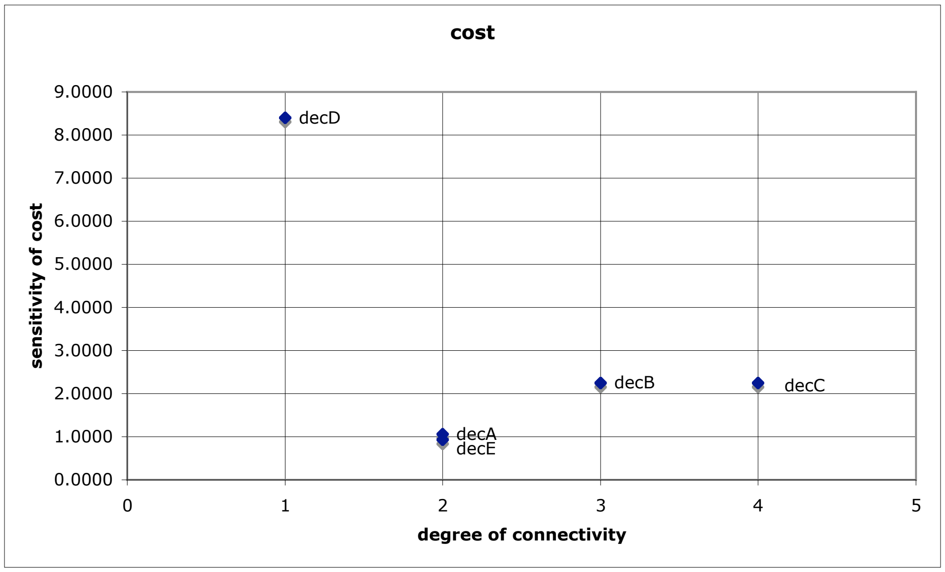


Figure 3-9: Decision space view for cost property.

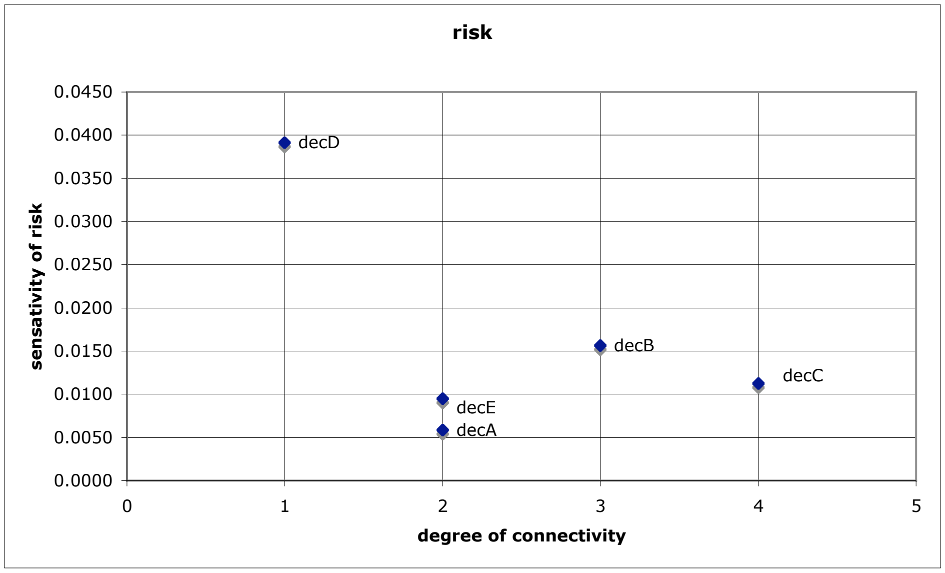


Figure 3-10: Decision space view for risk property.

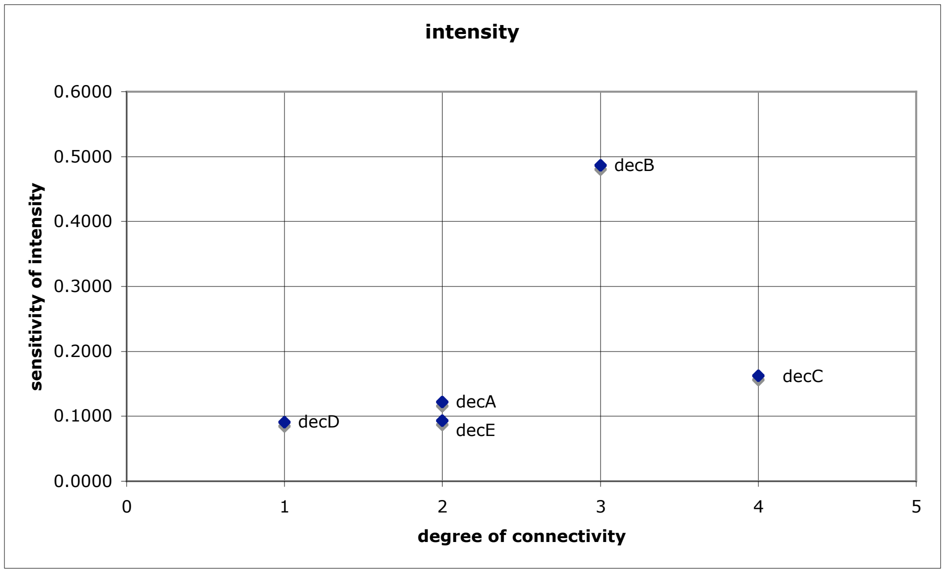


Figure 3-11: Decision space view for intensity property.

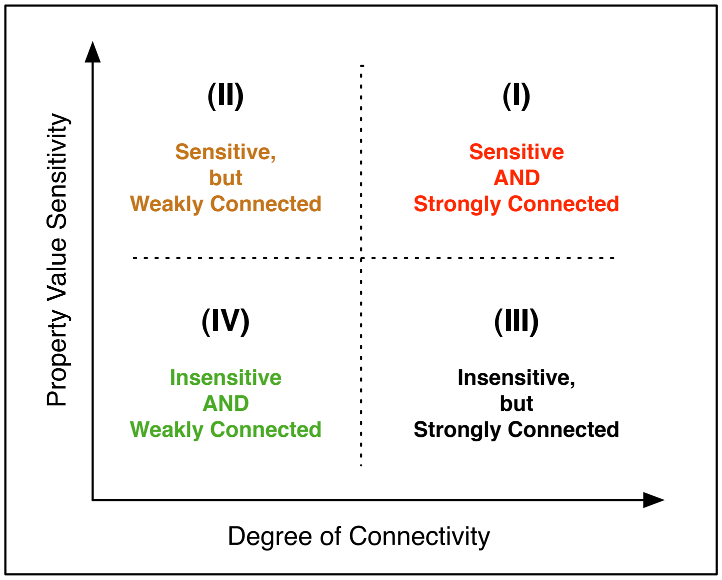


Figure 3-12: Decision space view – four quadrants.

A guide for the interpretation of the decision space view is show in Figure 3-12. The plot can be divided into four basic quadrants with which a decision-maker prioritize decisions. The four quadrants in Figure 3-12 are a qualitative measure of the different types of decisions in a systems architecture. It can be interpreted as a way to guide the order of "choice activity" in decision making. The quadrants can be interpreted as follows:

The upper right quadrant (I) contains decisions which are both sensitive and strongly connected. These decisions are connected to the highest number of other decisions and have the highest impact on the system properties. The decisions which lie in this quadrant for one or more of the properties should receive the most attention from the decision maker. It is likely that these decisions should be resolved early in the architecting process.

The upper left quadrant (II) constrains decisions which have a high measure of property variable sensitivity but are weakly connected to other decisions. A change in these decisions may influence system properties strongly, but will not influence many other decisions. The decisions that lie in this quadrant can be analyzed largely independently from the other decisions. Decisions in this category should be given the second highest priority.

The lower right quadrant (III) contains decisions which are do not strongly affect properties, but are strongly connected to other decisions. According to the property variable sensitivity, decisions in this quadrant are not likely to have a high impact on system properties. The decision maker can wait to resolve these decisions until the highly sensitive ones are resolved. The decision-maker should give the properties in this category the third highest priority. These decisions could be made late in the design phase, since they are more likely to be constrained by other decisions and do not have a strong impact on system properties.

The lower right quadrant (IV) contains decisions which are neither impactful in terms of property variable sensitivity nor in terms of connectivity to other decisions. The decisions in this quadrant could be left until the end of the decision-making process, since they are largely independent of other decisions. Decisions in this quadrant do not have a strong, direct effect on system properties or the set of feasible alternatives of other decisions.

It is important to note that the categorization of decisions in these quadrants is a first order estimate of how the decision variables should be prioritized. The two

Table 3-11: Decision Category Chart

Category	cost	risk	intensity
I: sensitive & strongly connected			decB
II: sensitive, but weakly connected	decD	decD	
III: insensitive & strongly connected	decB, decC	decB, decC	decC
IV: insensitive and weakly connected	decA, decE	decA, decE	decA, decD, decE

measures of impact, property variable sensitivity and degree of constraint, should not be considered precise, monotonic measures of how each decision in the complete set of decisions should be handled. They should be considered an estimate of how a systems architect could organize the in-depth analysis of these decisions.

The decisions in an ADG can be categorized in a decision category chart after simulation is complete. An example of this chart, which uses the results plotted in Figures 3-9, 3-10 and 3-11, is shown in Table 3-11.

3.6.2 Pareto Front View

A second way to view the simulation data is called a Pareto front plot. Pareto front plots are commonly used in the field of multi-objective optimization [PW00]. The *Pareto optimal set* is the set of non-dominated points in a set of data measured by at least two metrics[Par69, Par97]. In the case of the ADG methodology, the set of data is the set of all feasible combinations of decisions, $\mathcal{C}$. An element $c_i \in \mathcal{C}$ is considered a member of the Pareto set if there is no other $c_j \in \mathcal{C}$ such that c_j is better than c_i in terms of all properties. The points along a two-dimensional Pareto front can be interpreted as "the best performance for metric A, given that metric B is constant" or "the best performance for metric B, given that metric A is constant."

Pareto fronts are sometimes called the "efficient frontier". The term efficient is used to mean that for all points on the Pareto front, it is not possible to improve one metric of interest without simultaneously worsening at least one other metric of interest. Pareto fronts capture the notion that the set of "best" options for systems which are measured by properties which may be in opposition to each other can only be characterized as tradeoff between those properties.

Figure 3-13 contains three Pareto fronts for the example problem; cost vs. risk, cost vs. intensity, and risk vs. intensity. The U with a circle around it is the called the "utopia point". This point signifies the direction of the most desirable property values for the two property variables. The Pareto front is indicated by a red dashed line. All points which are not included in the set of points on the Pareto front are considered "dominated" points.

The set of feasible combinations of decisions identified using a Pareto front could be used as a way to select a subset of alternatives for decisions which result solutions "near" the Pareto front. A nearness metric that is appropriate for a given problem would have to be defined by the decision-maker. By doing this, the decision-maker could prune the decision space to remove sub-optimal alternatives before restarting the ADG analysis cycle (3-1). This is discussed further in Section 6.4.

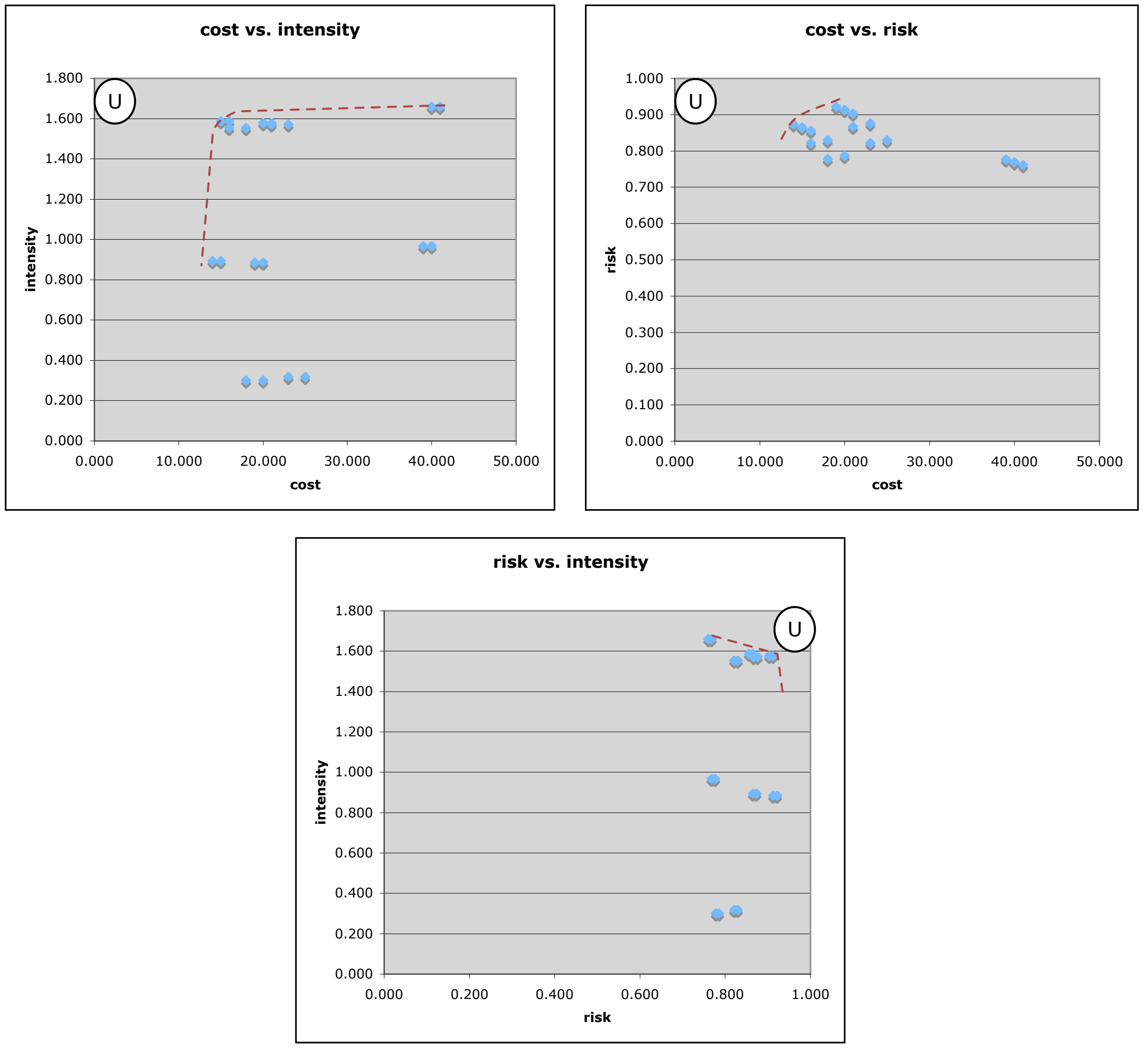


Figure 3-13: Pareto fronts for example problem. The points in the pareto plots correspond to the 20 feasible combinations of decision assignments in Table 3-9. The red dashed line is the pareto front. The U with a circle around it indicates the "utopia direction". Note that higher is better for the risk property.

3.6.3 Section Summary

This section presented two different ways to represent decision support data by viewing the data generated by simulating an ADG. The first view presented in this section was the decision space view. The decision space view is a two-dimensional scatter plot of the set decision variables which illustrates each variable's property variable sensitivity versus each variable's degree of connectivity. The decision space view can be used to partition the set of decision variables into four categories. These categorizes can be used to prioritize analysis of decision alternatives in an interconnected architectural decision problem.

The second view presented in this section was the Pareto front view. The Pareto front view is useful for determining which decisions lie in the non-dominated set. By analyzing a Pareto front view, decision alternatives that lead to dominated solutions can be identified, and potentially eliminated from the decision space.

3.7 Properties and Performance of the ADG Framework

This section analyses the properties and performance of the ADG methodology. There are two subsections. The first subsection discusses the properties of ADG simulation algorithm. The second subsection discusses the impact of the ADGsort1 and ADGsort2 structural reasoning algorithms on simulation performance.

3.7.1 Properties of ADG Simulation

ADG's simulation algorithm can be considered a search algorithm over a combinatorial space. According to Russell and Norvig[RN02], desirable properties of a search algorithm are:

- Property 1: A guarantee that the algorithm will terminate in finite time.
- Property 2: A guarantee that if a solution is returned by the algorithm, it is a valid solution.
- Property 3: A guarantee that if a valid solution exists, it will be returned by the algorithm.

These three properties can be guaranteed for ADG **only under the condition** that any property function in the ADG that is written in Jython also has these three properties.

The first property is a guarantee that the algorithm will terminate in finite time. ADG's simulation algorithm is guaranteed to terminate in finite time since OPNbuild1 (Algorithm 3.3) can only produce an OPN model which terminates in finite time. In general, OPN models do not necessarily terminate in finite time. Since OPN is a turing complete language, infinite cycles are permitted. However, Architecture Decision Graph uses the OPNbuild1 (Algorithm 3.3) to construct a model that is finite and acyclic. Since it is acyclic, all possible traces through the OPN model contain a finite number of object and processes. Each trace through the OPN model can be computed in a finite number of steps. Therefore, ADG's simulation algorithm is guaranteed to terminate.

The second property guarantees that when a solution is returned, it is a valid solution. ADG's simulation algorithm builds and executes an OPN model in order to return a set of the feasible combinations of decision assignments. An element in this set of decision assignments is considered valid **if and only if** it satisfies each and every logical constraint specified in the ADG. The OPN model compiled by OPNbuild1 (Algorithm 3.3) ensures that every element in this set is valid by embedding the logical constraints in post-conditions in such a way that all paths through the model to the final token collection object (called `__F`) must be checked against each and every logical constraint. Therefore, it is not possible to return a combination of decision assignments that is not valid. Therefore, ADG guarantees that when a solution is returned, it is a valid solution.

The third property guarantees that if a valid solution exists, then ADG's simulation algorithm will return that solution. Given sufficient computation resources, ADG has this property as well. As stated above, in the discussion of the second property, solutions are added to the set of feasible combinations of decision assignments **if and only if** they have been checked against each and every logical constraint. Furthermore, there are no other pre-conditions or post-conditions in the OPN model that might contain other constraints that could eliminate a token that contains a valid solution. Additionally, as discussed above, ADG is guaranteed to terminate. There are no operations in the algorithm that could prevent a valid solution from being returned. Therefore, a valid solution, if it exists, will be returned.

In summary, ADG has these three properties, under the conditions that all property functions written in Jython also have these three properties and there are sufficient computational resources available for simulation of the decision problem.

3.7.2 Impact of Structural Reasoning on Simulation Performance

The impact of the structural reasoning algorithms on the performance of ADG simulation can be measured by comparing them to performance of ADG simulation using all possible decision variable orderings. This was possible for the example problem in this Chapter, since its five decisions have 120 different possible variable orderings ($5! = 120$).

The results of the performance test are plotted in Figure 3-14. Note that there are two vertical axes in Figure 3-14: time measured in seconds and space measured in number of tokens. The results were sorted by number of generated tokens in each test run. This forced the blue curve corresponding to number of tokens to be monotonic.

Referring to Figure 3-14, the left vertical axis corresponds to time in seconds elapsed to reach a solution. The points measured by this axis are the red, square data points. Computational time was measured by executing the OPN model on an Apple Powerbook G4/1.5GHz/2GB running Mac OSX 10.4 and JVM 1.5. Execution time can only be measured in one second increments since this is the update cycle of the OPN token scheduler.

Referring to Figure 3-14, the right vertical axis corresponds to number of tokens generated. The points measured by this axis are the blue, diamond data points. The number of tokens generated is a proximate measure of memory consumption. The OPN token scheduler generates a new token to save its state history whenever any token is transformed by a process (see [Koo05] or Section 3.5.2 for details).

By doing a complete enumeration of all 120 possible variable orderings, this test demonstrates the performance of the heuristic ordering algorithms ADGsort1 and ADGsort2 *for this specific decision problem*. The performance of the OPN model when sorted by ADGsort1 and ADGsort2 is indicated by the green vertical lines in Figure 3-14.

The ordering produced by ADGsort1, {decC, decB, decA, decE, decD} , completed the simulation after generating 66 tokens in 3 seconds. Measured by number of tokens produced this was equal to eighth best possible variable ordering out all all 120 orderings in this test. The ordering generated by ADGSort2, {decC, decE, decA,

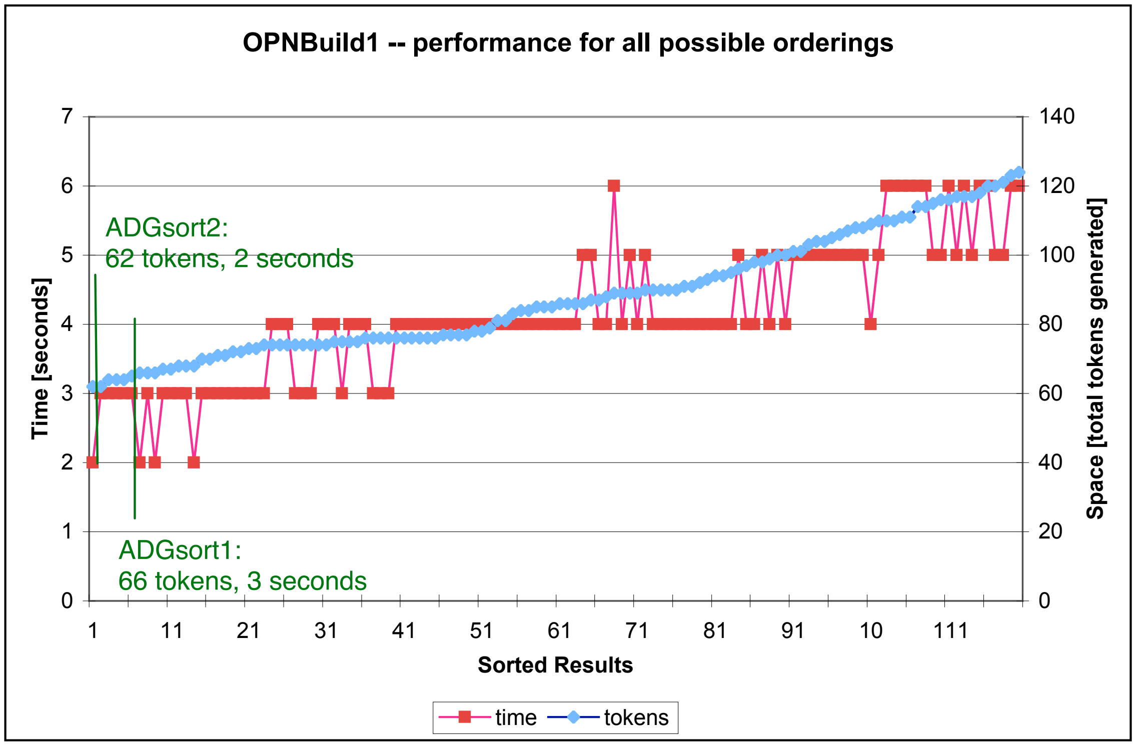


Figure 3-14: OPNBuild1 Performance. This is a 2-axis vertical plot, time [sec] and space [tokens] which compares all 120 possible orderings of the five decisions. The performance of the orderings produced by ADGsort1 and ADGsort2 are indicated with green vertical lines.

decB, decD}, completed the simulation after 62 tokens in 2 seconds of computational time. Measured by number of tokens produced, ADGsort2's order was equal to best possible variable ordering in this test out all 120 orderings in this test.

Since ADGsort1 and ADGsort2 are heuristic variable ordering algorithms, there is no guarantee of optimality of the order. However, in this case, both sorting routines produced decision variable orderings in the top 10% of all possible orderings, in terms off tokens genereated. (Additional tests of ADGsort1 and ADGsort2 are in Section 4.6.1 and Chapter 5.)

Worst Case Number of Tokens

Worst case upper bounds on the number of tokens generated by the OPN model can be calculated by ignoring tokens eliminated by the logical constraints. This subsection describes two equations for calculating the upper bounds on the number of generated tokens. The first equation does not take variable order into account. The second equation depends on decision variable order.

The worst case upper bound the number tokens generated by the OPN model produced by OPNbuild1 can be calculating using the following equation:

$$\text{number of tokens} \leq 1 + \sum_{k=1}^{n_D} |A_{max}|^k + \prod_{k=1}^{n_D} |A_k|, \tag{3.6}$$

where $|A_{max}|$ is the size of the largest set of alternatives for all decisions in D, n_D is the number of decision variables. The first term in the equation, 1, is the first token generated after executing the initialize process. The second term in the equation, $\sum_{k=1}^{n_D} |A_{max}|^k$ is the number of tokens generated by the part of the OPN model which evaluates combinations of decision variable assignments against the logical constraints to produce $\tilde{D}$. This term is an upper bound on the maximum number of tokens that could be generated if the logical constraints were suspended. The third term of the equation, $\prod_{k=1}^{n_D} |A_k|$, is the tokens generated by the metric calculation process, which could have as many tokens as the complete combinatorial space of the decision variables.

Using Equation 3.6, the upper bound on the number of generated tokens for the example problem is:

$$1 + (4 + 4^2 + 4^3 + 4^4 + 4^4 + 4^5) + (3 \times 4 \times 2 \times 2 \times 3 \times 2) = 1909 \text{ tokens}$$

Note that the actual worst case number of tokens generated was 122 tokens (See Figure 3-14).

A tighter upper bound on the number of tokens can be calculated when an ordered set of the decision variables in D is used. Instead of using $|A_{max}|$ to calculate the size of the combinatorial space, the actual size of each decision variables set of alternatives $|A_k|$ is used. The following equation is an upper bound for the number of tokens generated using an ordered set of decision variables:

$$\text{number of tokens} \leq 1 + \sum_{j=1}^{n_D} \left(\prod_{k=1}^{j} |A_k| \right) + \prod_{k=1}^{n_D} |A_k|, \tag{3.7}$$

where the integers $j = (1, .., n_D)$ are the indexes of an ordered set of the decision variables in D, and A_k is the set of alternatives for decision variable d_k. The second term on the right hand side of Equation 3.7, is an upper bound on the number of tokens generated by each branch of the OPN model. The upper bound on the sum of the tokens generated by each branch is reduced if the decision variables with a smaller number of alternatives are earlier in the order.

Using Equation 3.6, and the order from ADGsort1, the tighter upper bound on the number of generated tokens for the example problem is:

$$\begin{aligned}
&1+\\
&(2+2\times4+2\times4\times3+2\times4\times3\times2+2\times4\times3\times2\times3)+\\
&(3\times4\times2\times2\times3\times2)\\
&\quad=515 \text{ tokens}
\end{aligned}$$

Note that the actual number of tokens generated by the OPN using ADGsort1 as a variable ordering heuristic was 66 tokens, since some alternatives were eliminated by logical constraints.

3.8 Summary

ADG is a representation of an architectural candidate space. It explicit represents the candidate space as a graph of interconnected architectural decision variables, logical constraints, property variables, and property functions. It was designed with four assumptions: 1) the steps of architecture refinement can be characterized as decisions applied to the architecture space, 2) decision variables have a finite set of mutually exclusive alternatives, 3) continuous decision variables can be approximated using discrete variables, and 4) a decision-maker is not required to specify a decision variable order prior to analyzing the decision space.

Section 3.3 describes the representational primitives of ADG. They are designed to be explicit answers to questions that a decision-maker is faced with:

Table 3-12: Mapping of architecting questions to ADG's primitives.

Question	**ADG Primitives**
What are the decisions?	Decision Variables.
How will I evaluate them?	Property Variables.
How are they interconnected?	Logical Constraints and Property Functions.

Section 3.4 describes ADG's structural reasoning algorithms. ADG is intentionally designed to be an atemporal (declarative) representation of a decision problem. By doing this, the structure of a decision problem can be used to determine the order of decision variables rather than the requiring a decision-maker to specify the order manually. ADG provides two structural reasoning tools, ADGsort1 and ADGsort2, which determine a decision ordering based on degrees of connectivity and the size of the a decision variable's assignment domain. In practice, this works well for architectural decision problems, which usually do not follow a strict-time sequence.[2]

Section 3.5 describes how ADG uses the OPN kernel for enumerating and simulating feasible sets of decisions. Although OPN has been shown to be a powerful systems architecting tool in practice, it has also been challenging for systems architects to use effectively. Using ADG, an architect can explicitly model decision variables and connections, then automatically build an OPN model to enumerate and simulate feasible sets of decisions. The advantages of this are: 1) systems architects do not have to understand OPN's Petri-net-like environment, 2) ADG can automatically re-build a new, efficient model as the problem definition changes over time, and 3) although there is no guarantee of the efficiency ADG's structural reasoning algorithms, in all available test cases, it produces OPN executable models that are within the best 10% of all variable orders tested.[3]

[2]Some additional commentary on this issue is available in Chapter 4, where a mixed temporal and a-temporal problem is examined.

[3]additional test cases are shown in Chapter 4 and 5.

4 Apollo Architecture Study

4.1 Overview

This chapter presents a retrospective study of architectural decisions made for the Apollo lunar exploration program of the 1960's. This example of a systems architecture problem was chosen for three reasons. The first reason is that architecting the Apollo mission was and is still considered a "tough problem" (see R. Seamans, *Project Apollo: The Tough Decisions* [Sea05]). Arguably, the Apollo project to land a man on the Moon and return him safely to Earth was one of the most ambitious and unprecedented engineering challenges ever conceived. It is a benchmark problem in systems architecting, and an example of H. Simon's definition of "unprogrammed" decision-making (see Section 1.2).

The second reason for choosing this study is that historical evidence [Sea96, SKK05, BGS79, Han95, MC04, EMB$^+$78, Kin05, Hil04] shows that progress in engineering the Apollo program was limited until one critical decision was made: the so-called mission-mode decision. More than two years were spent iterating through many possible mission architectures until June 7, 1962. On that day, the decision to choose Lunar Orbit Rendezvous (LOR) (Figure 4-1) as the Apollo mission mode[1] over the other possible options set the whole program on a rapid path to the successful moon landing in 1969 [Hil04]. Selecting the mission-mode enabled the program to rapidly move forward with the detailed design and development of the spacecraft. The LOR decision is an excellent example of how identifying and making the most import decisions early in a systems architecting project can lead to the success of the program.

[1]A mission-mode is defined as the number, types, destinations, and interactions of vehicles for a space mission [SKC05a].

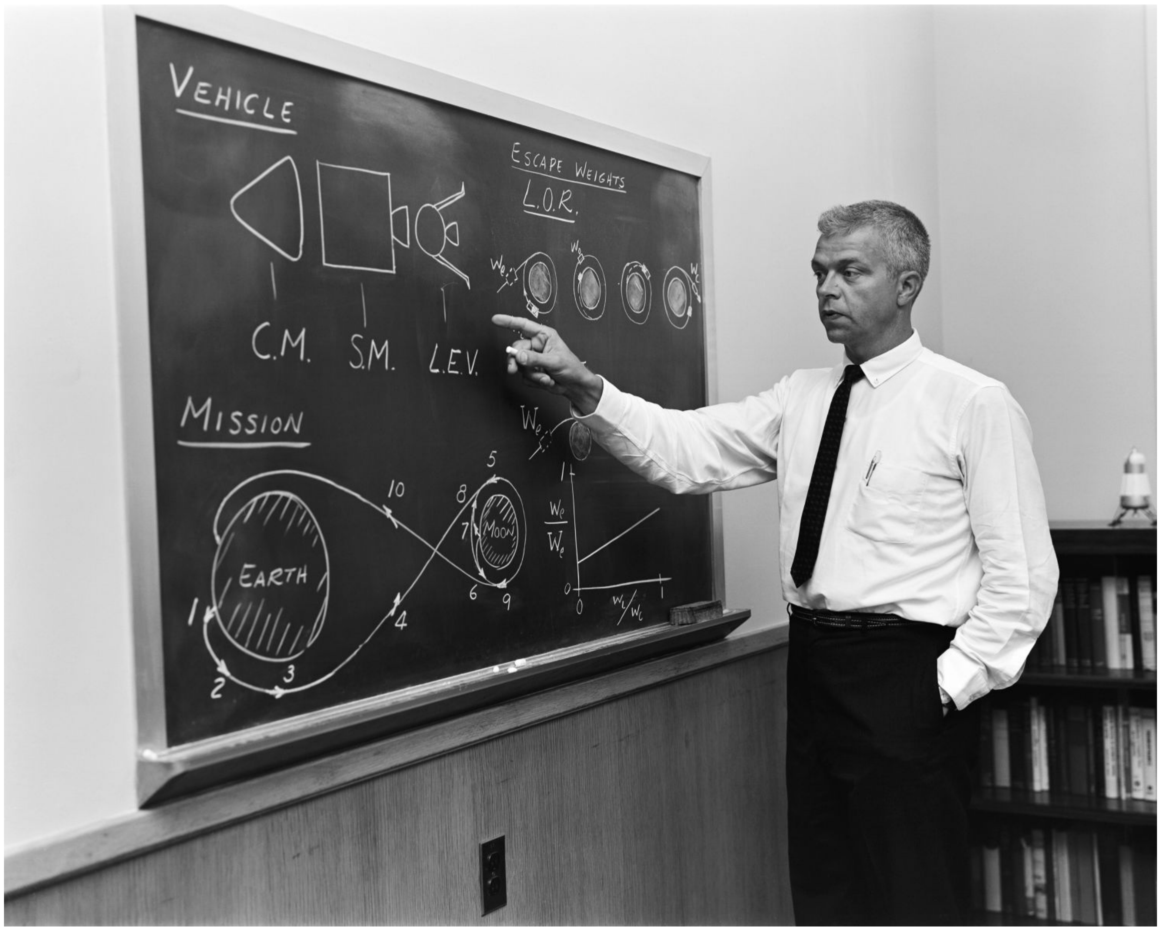


Figure 4-1: John C. Houbolt explaining the mission-mode known as Lunar Orbit Rendezvous (LOR). [credit: NASA photo]

The third reason for choosing the Apollo decision problem is that a comparison can be can be made between the ADG approach to studying the Apollo decisions and the motion language approach for studying the same problem used in Benjamin Koo's thesis [Koo05]. Koo used an earlier version of OPN to construct a decision model for Apollo in OPN's Petri-net-like structure of objects and processes.

The primary objective of this chapter is to demonstrate that ADG can be applied to a realistic architecting problem. Specifically, this chapter shows that ADG can be used to identify and prioritize decisions for the Apollo project using contemporary information from the early 1960's. A secondary objective of this chapter is to present guidance for a systems architect using ADG. Section 4.2.1 provides four general guiding principles which can be used to transform an architecting problem into a decision problem.

The outline of this chapter mirrors the outline of Chapter 3. The four processes of the architecting decision graph cycle (Figure 3-1) are addressed in order: Representing, Structural Reasoning, Simulating, and Viewing. The chapter concludes with three discussions: A discussion of the impact of the structural reasoning algorithms on the simulation performance, a discussion comparing Koo's approach to the ADG approach, and a summary of the findings of this chapter.

4.2 Representing

Referring to the ADG cycle presented in Figure 3-1, the first process is representing available knowledge about the system by encoding it using an ADG. In order to simplify the initial presentation of ADG, the discussion of the representing process in Chapter 3 only briefly describes the practical issues involved in developing an ADG model for a real architecting problem. This chapter addresses that issue in more depth by first presenting four general principles for representing an architectural space as a set of decisions. These principles are applied to the Apollo architecting task.

4.2.1 General Principles for Formulating a Decision Problem using ADG

This subsection presents general guidelines for modeling an architectural problem using ADG. Four guiding principles for developing the set of decision variables representing an architectural configuration space are presented below. These four principles were developed by leveraging general systems architecting principles [Cra07, Rec91, MR02], the guidance outlined in previous research using Object Process Network (see Section 2.3.4) [SKC05a, SKC06, SKC07], as well as experience using ADG as an architecting tool.

1) Set the Boundaries: The first guiding principle is to *set the boundaries of the architectural space under consideration.* This principle is generally applicable to any enumerative method which searches a design space because it is impossible to comprehensively search and evaluate an infinitely large configuration space of architectures within finite time. In order to complete the analysis without exceeding the amount of available computational resources, reasonable bounds must be set on the space of architectures which will be considered.

Analysis of an architectural space using ADG requires that the decision-maker or architect have at least some knowledge about the system before any decision model can be developed. The task of pre-selection of categories of architectures under consideration necessarily relies on the judgement of the architect. It requires a preliminary understanding of what types of architectures would be reasonable and which ones would be unreasonable.

As a rule of thumb, it is often advisable to start formulating the problem using "tight" boundaries by only considering a limited set of alternatives in early studies. In successive studies, these boundaries can be "loosened" to allow the consideration of additional architectures.

2) Decision Variables Should be Inputs to Property Functions: The second guiding principle is to *develop the set of property functions for evaluating the system of interest before trying to develop the set of the decision variables.* Experience has shown that by first developing the property functions that would be needed to evaluate the system, the decisions variables that are needed as inputs for these property functions become more evident.

The roots of the second principle can be found in the systems architecting literature. It is generally accepted that the "value proposition" or "value equation" for a system is a relatively stable, high-level abstraction for a system [Rec91, Cra07]. By focusing attention on how a system delivers value, the set of elements of that system which contribute to delivering value can be identified. For ADG analysis, the "value proposition" is the ability to calculate the properties of interest. The elements of the system that enable value delivery are the set of decisions which are inputs for the property functions. The basic modeling assumption is that property functions are used to measure value delivery, and hence decision variables that are inputs to property functions also directly influence value delivery.

3) Capture Architecturally Distinguishing Decisions: The third guiding principle for formulating a decision problem using ADG is to *capture the architecturally distinguishing decisions.* An architecturally distinguishing decision is one that potentially changes the overall high-level concept of the architecture to be implemented. For example, when designing a road vehicle, an architecturally distinguishing decision is the decision of "how many wheels?" with the set of alternatives {2,4}. By choosing two wheels, the concept becomes motorcycle-like. The architecture must take into consideration that the vehicle must be balanced. By choosing four wheels the concept becomes more car-like. Assuming the four wheels are not in a single line, balancing the vehicle does not become a major design issue. Cars and motorcycles have the same value delivering function: transporting people and cargo. However, their architectural concepts, the mapping of function to form[Cra07], are different.

4) Keep it Simple: The fourth guiding principle is to *keep the problem formulation as simple as possible, but no simpler.* This principle is often attributed to Albert Einstein [HKG01], however the idea of striving to keep a model simple probably predates Einstein [Mae06]. Since ADG is an enumerative procedure, this principle is important. ADG's simulation algorithm for exploring the decision space consumes computational resources at a rate that scales exponentially with the number of decision variables (see Section 3.7.2). Since computational resources are limited, there is an upper limit to the the size of a decision problem that ADG can explore in a reasonable amount of time. To follow this principle, the decisions in the ADG should be limited to ones that are architecturally distinguishing or are inputs to metrics functions. In general, decision variables which are neither inputs to property functions nor architecturally distinguishing should not be included in the model.

4.2.2 Formulating the Apollo Architecture Decision Graph

In general, the four guiding principles presented in the preceding subsection are applied iteratively to an architecture problem to develop a set of decision variables, logical constraints, property variables, and property functions. This subsection describes how the four principles were specifically applied to the Apollo architecture problem.

The first guiding principle for formulating the decision problem is to set the boundaries of the architectural space under consideration. The highest level specification of the system can be derived from President Kennedy's 1961 speech on the Urgent Needs of the Nation: Value was delivered by the Apollo system if it satisfied the goal to improve the international influence and to increase the prestige of the United States by "landing a man on the moon and returning him safely to the Earth [before the end of the 1960's]"[Ken61]. This statement sets the requirement that the Apollo project should land at least one man on the moon and return him to Earth. Clearly, any architecture which do not provide a mission concept with the means to reach this goal should not be included within the bounds of the architectural decision space.

In the early 1960's, the idea of landing one man on the moon was considered an extremely ambitious goal since the U.S. space program had only successfully launched one astronaut into sub-orbital space at the time. However, since a lunar mission is a relatively long, and complex space mission, it may have been feasible, but risky to send one lone astronaut on the voyage. Since the main objective of the project was to develop an architecture that could complete the engineering goals successfully by the end of decade, it is not necessary to consider missions with large numbers of astronauts, such as von Braun's *Conquest of the Moon* [vB53]. In this study we bound the number of crew members to at least one and no more than three.

The feasibility and reliability of in-space rendezvous and docking was a heavily debated topic in the early planning years of the Apollo project. John C. Houbolt showed through extensive analysis that is was challenging, but technically feasible. He argued that missions including rendezvous, especially rendezvous and docking in lunar orbit, should be considered [Han99, Hou61b, Hou61a]. The alternatives for mission modes which include rendezvous and docking are included within the bounds of the Apollo decision model.

The second principle states that the decision variables should be inputs to the property functions. Two property functions that are considered important in the development of any space mission are the total mass of the mission and the probability of mission success [Wer99]. The calculation for the total mass of the mission must take into account the mission mode, the crew size, as well as the fuel types to be used for spacecraft maneuvers. The second property function, the calculation of the probability of success, depends on the maneuvers included in the mission mode, the crew size, and the fuel types. The methodology for calculating these two properties will be discussed in Section 4.2.5.

The third principle states that the decision model should include the architecturally distinguishing decisions. Among the decisions for Apollo, decision variables related to the mission mode are clear examples of architecturally distinguishing decisions. For example, if the mission mode includes lunar orbit rendezvous, the concept for the mission includes two vehicles: one orbiting crew vehicle which has a heat shield so that it can re-enter the Earth's atmosphere, and a lunar lander vehicle that is specialized for descent to the surface of the moon and ascent back to lunar orbit. A change in this decision would imply a change to the assignment of the architecture's functions to its forms. Therefore, it is important to include these decisions in the decision model. The other two categories of decisions that were mentioned, the crew size and fuel types, are not architecturally distinguishing, however, they are important for calculating the two property functions, so they must also be included.

The fourth principle states the model should be kept as simple as possible, but no simpler. The application of this principle lead to the removal other decision variables, like the launch site decision and the task assignments to the different NASA centers. Neither of these decisions has a direct impact on the total mission mass or the probability of mission success. More subtly, neither one of these decisions is architecturally distinguishing, because neither of them changes the mapping of function to form in the *high-level concept* the architecture. A change in these decisions could change the implementation of the architecture, but would not change the feasibility of particular mission modes, crew sizes, or fuel types.

4.2.3 Apollo Decision Variables

After considering the four principles and several iterations building and testing the ADG model of the Apollo architecture decision space, a set of nine decision variables were selected for this study. The set of decision variables includes decision variables related to the mission mode, the crew size, and the rocket fuel types used for Apollo. Table 4-1 is a morphological matrix (see Section 2.3.1) of these nine decisions.

Table 4-1: The set of nine decision variables for the Apollo study.

shortID	Decision	units	alt A	alt B	alt C	alt D
EOR	Earth Orbit Rendezvous	none	no	yes		
earthLaunch	Earth Launch Type	none	orbit	direct		
LOR	Lunar Orbit Rendezvous	none	no	yes		
moonArrival	Arrival at Moon	none	orbit	direct		
moonDeparture	Departure from Moon	none	orbit	direct		
cmCrew	Command Module Crew	people	2	3		
lmCrew	Lunar Module Crew	people	0	1	2	3
smFuel	service module fuel	none	cryogenic	storable		
lmFuel	lunar module fuel	none	NA	cryogenic	storable	

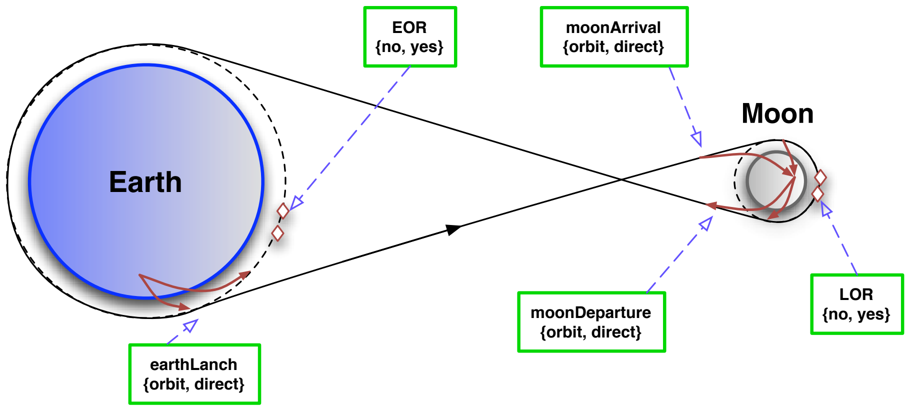


Figure 4-2: The five mission-mode related decision variables.

Five of the decisions are mission-mode related decisions: EOR, earthLaunch, LOR, moonArrival, and moonDeparture. The mission mode-related decisions are graphically indicated in Figure 4-2. All five of these decision variables indicate a choice of alternative maneuvers at different points of the mission. By combining one alternative from each of the five decisions, a mission-mode can be defined. For example, during the Apollo architecting process, historical records indicate that three different classes of mission modes were under consideration: Direct, Earth Orbit Rendezvous, and Lunar Orbit Rendezvous (see Figure 4-3). Table 4-2 shows how the three Apollo mission modes can be mapped to the set of five decision variables.

Table 4-2: Mapping of historical Apollo mission modes (Figure 4-3) to the nine ADG decision variables. Note that the combinations of decision variable assignments listed in right column must also satisfy the logical constraints in Section 4.2.4.

Apollo Mission Mode	**ADG Decisions**
Direct Mission Mode:	EOR is no, earth Launch is orbit or direct, moonArrival is orbit or direct, LOR is no, and moonDeparture is orbit or direct.
EOR Mission Mode:	EOR is yes, earth Launch is orbit, moonArrival is orbit or direct, LOR is no or yes, and moonDeparture is orbit or direct.
LOR Mission Mode:	EOR is no or yes, earth Launch is orbit or direct, moonArrival is orbit, LOR is yes, and moonDeparture is orbit

In addition to these three major classes of mission-modes, this set of five decision variables can be used to specify combinations and variations of these three, such as a mission-mode which has maneuvers for Earth orbit rendezvous and docking as well as lunar orbit rendezvous and docking. In this study, this mission-mode is named "EOR+LOR".

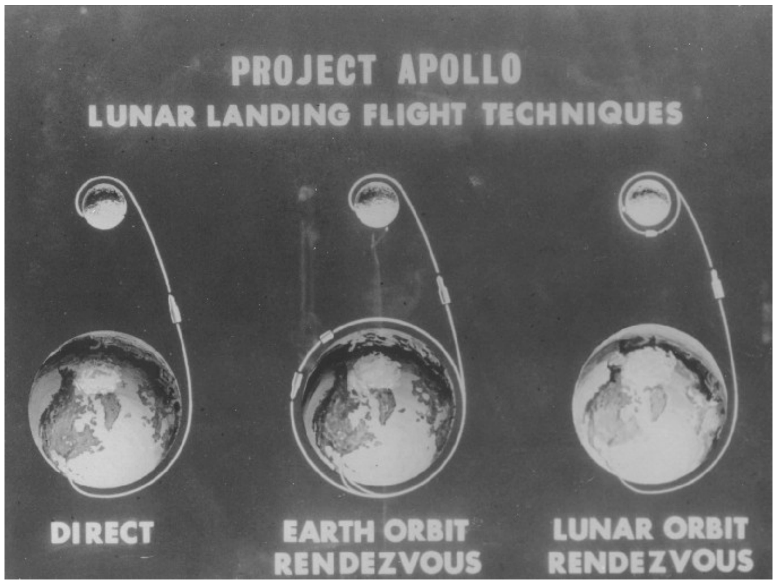


Figure 4-3: Mission modes under consideration during the Apollo Program: Direct, EOR, and LOR. [Source: NASA]

The four remaining decisions are related to crew sizes and fuel types: cmCrew, lmCrew, smFuel, and lmFuel. These four decisions, as well as the five decisions related to the mission-mode are explained in the following list:

EOR : Earth Orbit Rendezvous, {no, yes}. The Earth orbit rendezvous decision specifies whether or not the spacecraft rendezvous and docks with a second spacecraft after launch.

earthLaunch : Earth Launch Type, {orbit, direct}. This decision specifies whether the spacecraft enters Earth orbit after launch or is directly injected towards the moon.

LOR : Lunar Orbit Rendezvous, {no, yes}. The lunar orbit rendezvous decision specifies whether or not there is a second lander spacecraft which lands on the moon. The alternative "yes" implies that there is a second lander spacecraft. The mission mode must then include entering lunar orbit, undocking the second spacecraft, descending to the lunar surface, ascending from the lunar surface, then rendezvous and docking with the first spacecraft in lunar orbit. The alternative "no" indicates that lunar orbit rendezvous does not occur, and by implication there is only one crewed spacecraft which transports the crew to the lunar surface and back to Earth.

moonArrival : Arrival at Moon, {orbit, direct}. The arrival at the moon decision specifies if the spacecraft enters lunar orbit, or directly descends to the lunar surface.

moonDeparture : Departure from Moon, {orbit, direct}. The departure from the moon decision specifies if the spacecraft enters lunar orbit after launching from the moon, or is directly injected into a trans-Earth trajectory.

cmCrew : Command Module Crew, {2, 3}. This decision specifies the command module crew size at launch

lmCrew : Lunar Module Crew, {0,1,2,3}. This decision specifies the lunar module crew size. The alternative 0 is used to indicate that there is no lunar module crew in the case that there is no separate lunar lander.

smFuel : Service Module Fuel Type, {cryogenic, storable}. This decision specifies one of two types of service module rocket fuel. Cryogenic is LOX/H2 propellant, and storable is a hypergolic propellant [Wer99].

lmFuel : Lunar Module Fuel Type, {NA, cryogenic, storable}. This decision specifies the type of lunar lander rocket fuel. Cryogenic is LOX/H2 propellant, and storable is a hypergolic propellant. The assignment NA is used in the case that there is no separate lunar module and therefore this decision variable is not applicable.

4.2.4 Apollo Logical Constraints

The constraints are generated by capturing available knowledge about the system and the relationships between the decision variables. The list of constraints is in Table 4-3.

Table 4-3: Apollo Logical Constraints.

name	scope	equation
EORconstraint	EOR,earthLaunch	(EOR == yes && earthLaunch == orbit) || (EOR == no)
LORconstraint	LOR,moonArrival	(LOR == yes && moonArrival == orbit) || (LOR == no)
moonLeaving	LOR,moonDeparture	(LOR == yes && moonDeparture == orbit) || (LOR == no)
lmcmcrew	cmCrew,lmCrew	(cmCrew >= lmCrew)
lmexists	LOR,lmCrew	(LOR == no && lmCrew == 0) || (LOR == yes && lmCrew >0)
lmFuelConstraint	LOR,lmFuel	(LOR == no && lmFuel == NA) || (LOR == yes && lmFuel != NA)

Table 4-4: Apollo Logical Constraint Tables.

EORConstraint	EOR	
earthLaunch	no	yes
orbit	1	1
direct	1	0

lmcmcrew	cmCrew	
lmCrew	2	3
0	1	1
1	1	1
2	1	1
3	0	1

LORConstraint	LOR	
moonArrival	no	yes
orbit	1	1
direct	1	0

lmexists	LOR	
lmCrew	no	yes
0	1	0
1	0	1
2	0	1
3	0	1

moonLeaving	LOR	
moonDeparture	no	yes
orbit	1	1
direct	1	0

lmFuelConstraint	LOR	
lmFuel	no	yes
NA	1	0
cryogenic	0	1
storable	0	1

Each constraint is explained in the list below. Constraint tables for each constraint are shown in Table 4-4.

EORconstraint scope: EOR, earthLaunch. If there is an Earth orbit rendezvous, then this implies that the earthLaunch decision must be equal to orbit, since it's impossible to rendezvous without entering Earth orbit first.

LORconstraint. scope: LOR, moonArrival. If there is a lunar orbit rendezvous in the mission mode, this implies that the moonArrival Decision must be equal to orbit, since it's impossible to complete the rendezvous maneuver without entering lunar orbit before descending to the lunar surface.

moonLeaving. scope: LOR, moonDeparture. If there is a lunar orbit rendezvous in the mission mode, this implies that the moonDeparture Decision must be equal to orbit, since it's impossible to complete the rendezvous maneuver without entering lunar orbit after ascending from the lunar surface.

lmcmcrew. scope: lmcrew, cmcrew. This constraint restricts the crew size of the lunar module to be less than or equal to the crew size of the command module.

lmexists. scope: lmCrew, LOR. This constraint forces lmCrew to be zero if there is no lunar orbit rendezvous.

lmFuelConstraint. scope: lmFuel, LOR. This constraint forces lmFuel to be NA if there is no lunar orbit rendezvous.

4.2.5 Apollo Property Variables and Property Functions

As explained above, there are two metrics that are especially useful in predicting the success of the Apollo project: operational risk, and initial mass to low Earth orbit (IMLEO). These two properties are outlined in next two subsections.

Initial Mass to Low Earth Orbit (IMLEO) Property

Following common aerospace engineering practice, the rocket equation[Cho96] can be used to estimate vehicle masses, depending on the velocity increment they must supply, their payload and the mission mode. The equation is defined as follows:

$$m_f = m_i \exp\left(\frac{-\Delta V}{g_0 \cdot I_{sp}}\right), \tag{4.1}$$

where ΔV is the difference in velocity over the entire period of the maneuver, g_0 is the gravitational constant, I_{sp} is the specific impulse of the propulsion system, m_f is the final mass after the maneuver, and m_i is the initial mass before the maneuver. The two mass terms m_f and m_i can be broken down as follows:

$$m_f = m_{bo} + m_{pl} \tag{4.2}$$

$$m_i = m_{bo} + m_{pl} + m_{prop}, \tag{4.3}$$

where m_{bo} is the burnout (structure-only) mass, m_{pl} is the payload mass, and m_{prop} is the propellant (fuel) mass. For a multi-stage rocket system, the rocket equation can be applied recursively for each maneuver. If the "payload" of a stage is actually another rocket with its own fuel and payload, then m_{pl} becomes the initial mass for the next application of the equation.

In this study, values for constants such as the structural mass ratios, propulsion characteristics, and models for crew compartment sizes were taken from a combination of historic data and the assumptions used in the contemporary 1961 Houbolt Report [Hou61a].

Risk Property

In addition to IMLEO, the other major factor in selection of the mission mode was operational risk. In order to calculate operational risk we extended the methodology presented in Koo's Apollo study [Koo05]. The risk property variable for a complete set of decision variables that have been assigned particular alternatives is calculated by multiplying the entries in the table using the following equation:

$$risk_{total} = \prod_{i=1}^{n} risk(d_i) \tag{4.4}$$

where the decision variables are labeled from 1 to n and $risk(d_i)$ is a the risk multiplier for a particular decision variable assignment derived from Table 4-5. For example, the risk multiplier for $risk(EOR = yes)$ is 0.95, according the table.

Note that the risk property is non-separable since the risk due to the cryogenic fuel for service module depends on the number of burns performed by that module's engine. The number of burns depends on the LOR decision variable. If LOR does not exist, the service module's engine must fire as many as four times: once for lunar orbit insertion, once for decent, once for ascent, and once for lunar orbit departure. If the architecture does not include LOR, the service module may burn as few as two times: once for lunar orbit insertion and once for lunar orbit departure. This abnormality in the risk table is noted under the entry for service module fuel (smFuel). The risk due to the service module fuel type has an exponent equal to the number of service module burns. Since this method for calculating the risk property does not qualify as a separable property function (see Section 3.3.1), it was implemented in the ADG as a Jython function. The code listing for the Jython functions for both risk and IMLEO are included in Appendix A.

Table 4-5: Risk Property Table.

CAT	shortID	Decision	units	alt A	alt B	alt C	alt D
decision	EOR	Earth Orbit Rendezvous	none	no	yes		
prop	risk			0.98	0.95		
decision	earthLaunch	Earth Launch Type	none	orbit	direct		
prop	risk			0.99	0.9		
decision	LOR	Lunar Orbit Rendezvous	none	no	yes		
prop	risk			1	0.95		
decision	moonArrival	Arrival at Moon	none	orbit	direct		
prop	risk			0.99	0.95		
decision	moonDeparture	Departure from Moon	none	orbit	direct		
prop	risk			0.9	0.9		
decision	cmCrew	Command Module Crew	people	2	3		
prop	risk			1	1		
decision	lmCrew	Lunar Module Crew	people	0	1	2	3
prop	risk			1	0.9	1	1
decision	smFuel	service module fuel	none	cryogenic	storable		
prop	risk			.95^(burns)	1		
decision	lmFuel	lunar module fuel	none	NA	cryogenic	storable	
prop	risk			1	0.9025	1	

The risk property can be interpreted as a relative measure of the probability of mission success. In NASA terminology, this risk measure is equal to $1 - LOM$, where 1 indicates the best case (100% probability of success) and LOM is the probability of loss of mission.

4.2.6 Apollo Architecture Decision Graph

Figure 4-4 is the complete view of the Architecture Decision Graph (ADG) for the the Apollo architecture decision problem. The complete view provides a visualization of the connectivity of the decision variables, logical constraints, property variables, and property functions. Since it is visually difficult to pick out the important features of this graph from the complete view alone, the two additional views of the ADG described in Section 3.3.2, the logical view and the properties view, are presented in Figures 4-5 and 4-6.

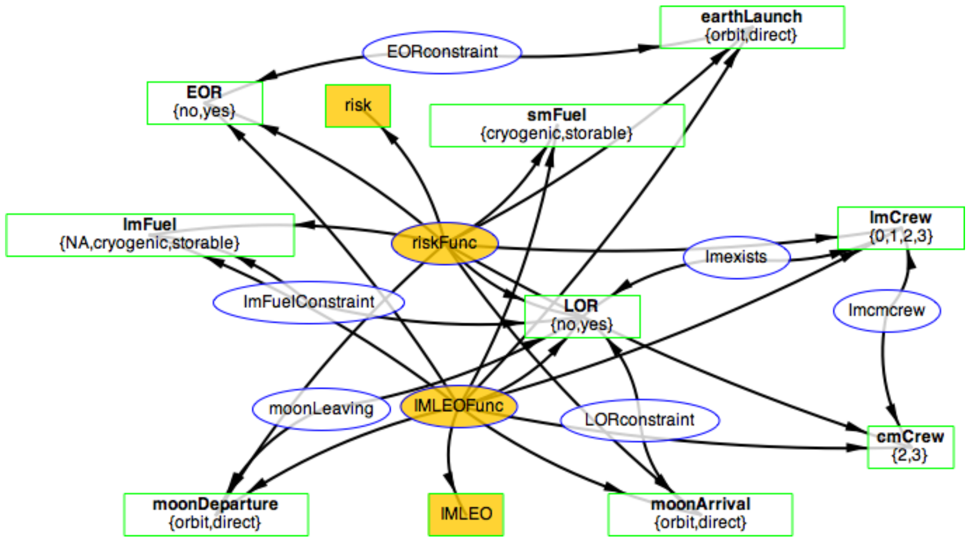


Figure 4-4: Apollo ADG – Complete View. The decision variables are green rectangles with a clear background. Each decision variable's set of alternatives is listed in braces below the decision variable's name. The property variables are green rectangles with an orange background. The logical constraints are blue ellipses with clear background, and the property variables are blue ellipses with an orange background.

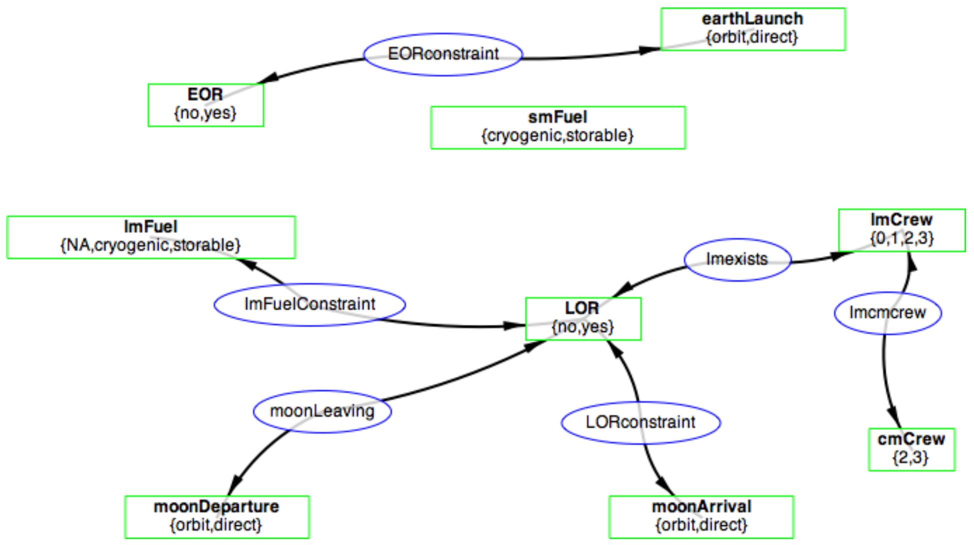


Figure 4-5: Apollo ADG – Logical View. This is an alternative visualization of the ADG. In this view the property variables and property functions have been removed. Only the decision variables and logical constraints appear. This view is useful for visualizing only the logical constraint relations between decisions variables.

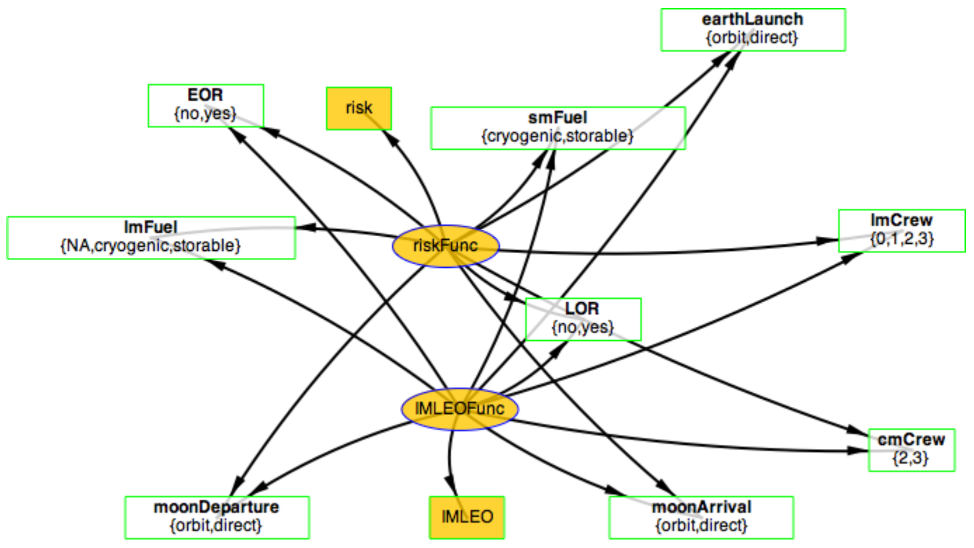


Figure 4-6: Apollo ADG – Properties View. This is another alternative visualization of the ADG. In this view the logical constraints relations have been removed. Three types of nodes remain: property variables, property functions, and decision variables. This view is useful for visualizing only the structure of the property value calculations.

Figure 4-5 is the logical view. In this view, certain features of the decision problem become apparent. By inspecting the connectivity of the graph, it is clear that decisions about maneuvers near the moon and the configuration of the lunar module (LOR, lmFuel, lmCrew, and moonDeparture) are not logically connected to the decisions related to maneuvers near Earth (EOR and earthLaunch) nor the decision about the service module fuel type (smFuel). These types of architectural features are not immediately apparent through reading the textual and tabular description of the decision problem. However, they become apparent though the use of the logical view.

Figure 4-6 is the properties view. In this view it becomes apparent that the three logically unconnected parts of the problem are linked in terms of system properties. In this ADG both the risk function and the IMLEO function are connected to all of the decision variables. Although the decision variables can be split into three logically unrelated sets, they all must be taken into consideration when calculating the property functions.

From a systems engineering management perspective, the knowledge gained about the decision variables through the visual inspection of these three views can inform engineering task breakdown and scheduling. This impact will be discussed further in Sections 4.3 and 4.7.

4.3 Structural Reasoning

The structural reasoning process is the second process in the ADG cycle (Figure 3-1). It involves analyzing the structure of the ADG and producing structural information about the decision problem. Note that some structural reasoning results were already discussed in the previous section. In that section, information about the structure of the problem was gained through visual inspection of the ADG alone. This section formalizes that notion by applying the two structural reasoning algorithms introduced in Section 3.4 to the Apollo ADG.

The structural reasoning results of two algorithms from Section 3.4, ADGsort1 and ADGsort2, are presented in Table 4-6. ADGsort1 calculates the degree of connectivity of the decision variables. This is a first order measure of the how many decisions are directly connected through adjacent logical constraints. ADGsort2 is an alternative measure of the structural properties of the ADG which adjusts the degree of connectivity by subtracting the size of each decision variable's set of alternatives.

Table 4-6: Results of ADGsort1 and ADGsort2 for the Apollo decision problem. ADGsort1 calculates the degree of connectivity and ADGsort2 calculates the sort2rank.

decision variable	degree of connectivity	sort2rank
EOR	1	-1
earthLaunch	1	-1
LOR	4	2
moonArrival	1	-1
moonDeparture	1	-1
cmCrew	1	-1
lmCrew	2	-2
smFuel	0	-2
lmFuel	1	-2

Specific descriptions of these algorithms are contained in Section 3.4.

The primary purpose of applying the structural reasoning algorithms to the ADG is to provide a heuristic to enhance simulation performance. The values for either degree of connectivity or sort2rank are used to guide the construction of an executable OPN model. In Section 4.4, the OPN model is built according to the ordering specified by sort2rank. The impact of both structural reasoning results on the efficiency of decision problem simulation is discussed in Section 4.6.1.

From a systems engineering management perspective, the results can also be used for task prioritization and scheduling. A high ranking in terms of degree of connectivity indicates that a decision is highly connected. In this example, the LOR decision variable is connected to four other decisions through logical constraints (degree of connectivity=4). This indicates that a change in the LOR decision variable may impact the set of available alternatives for at least four other decision variables (lmFuel, lmCrew, moonDeparture, and moonArrival). On the other extreme, the service module fuel decision variable (smFuel) is not logically connected to other decision variables (degree of connectivity=0). A change in service module fuel type does not change the set of available alternatives for any other decisions.

This information derived from the degree of connectivity is useful to an engineering manager. It implies that a team considering the LOR decision variable must collaborate with any other teams considering the four other connected decision variables. On the other extreme, a team considering the smFuel decision variable can work independently. Although changing the selection of smFuel may impact the system-wide properties (risk and IMLEO), it does not impact the feasibility of alternatives of other decision variables. Of course, the validity of these task management implications depends on the validity of the model itself. The aim of this approach is to enhance the judgement of a systems architect by providing guidance for the management of architecting tasks by identifying features in the structure of the problem.

4.4 Simulating

The simulating process is the third process in the ADG cycle (Figure 3-1). It involves transforming the structural information about the ADG into an executable OPN model and then using the OPN model to produce a set of feasible combinations of decision variable assignments and their properties, $\mathcal{C}$. The algorithm for completing this task, OPNbuild1, is described in detail in Section 3.5. OPNbuild1 requires a decision variable ordering as one of its inputs. For the Apollo decision problem, the results of ADGsort2 were used (Table 4-6). The resulting OPN executable model is shown in Figure 4-7.

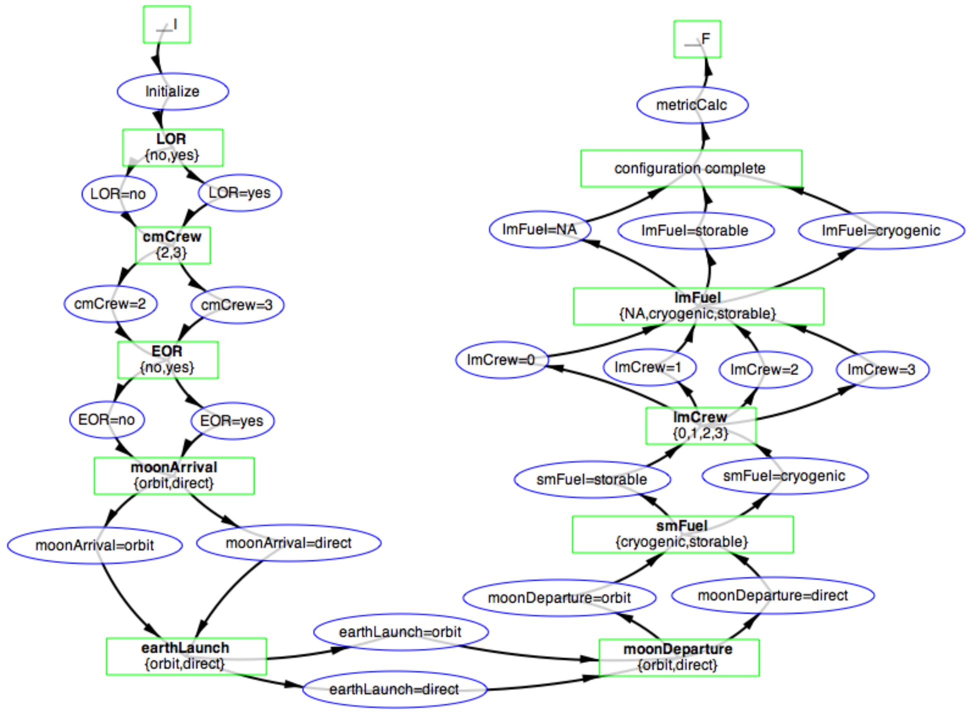


Figure 4-7: Apollo OPN executable model. Produced by the OPNBuild1 algorithm using the decision variable order resulting from the ADGsort2 algorithm.

Running the OPN executable model in Figure 4-7 produces 138 feasible combinations of decisions out of a total combinatorial space consisting of 1536 combinations of decision variable assignments.[2] The results of the simulation are presented in the next section. Discussion of the performance of the simulation algorithm is in Section 4.6.1.

4.5 Viewing

The fourth process of the ADG cycle (Figure 3-1) is decision data viewing. The viewing process transforms the feasible combinations of decision variable assignments and their properties, $\mathcal{C}$, into new available knowledge for the decision-maker. Section 3.6 describes two methods for viewing decision data: Pareto front plots and decision space views. Both of these methods were applied to the Apollo architecting problem and the results are presented below.

4.5.1 Pareto Front Plot

The Pareto front plot (see Section 3.6.2) shown in Figure 4-8 contains some interesting results. Among the solutions on the Pareto front are the competing mission modes for the Apollo program as well as a point which matches the configuration of the Soviet lunar mission architecture.

Points 1 and 2 are “direct” missions with three and two crew members, respectively. A direct mission mode implies that the mission has neither lunar orbit rendezvous nor Earth orbit rendezvous. These types of missions were among the ones initially supported Wernher von Braun’s group at Marshall Space Flight Center [Han99, Han95]. In the direct missions, the lunar module does not exist, and therefore there is no lunar module crew. Points 3, 4, 5, 6, and 8 are architectures which include lunar orbit rendezvous maneuvers. Point 3 matches the configuration of Apollo: It has three crew members in the command module, two crew members in the lunar module and uses storable propellants for both the service module and the lunar module [MC04]. Point 8, which is the minimum mass configuration, uses two crew members in a command module and one crew member in a lander with cryogenic propellants. Point 8 is the point which most closely models the proposed Soviet lunar mission’s architecture [Sid03, Har99, Wil89]. Point 7 is the only EOR mission which appears on the Pareto front. It has a configuration of two crew in the command module, and uses cryogenic propellants.

[2]The total size of the combinatorial space is calculated by suspending the logical constraints.

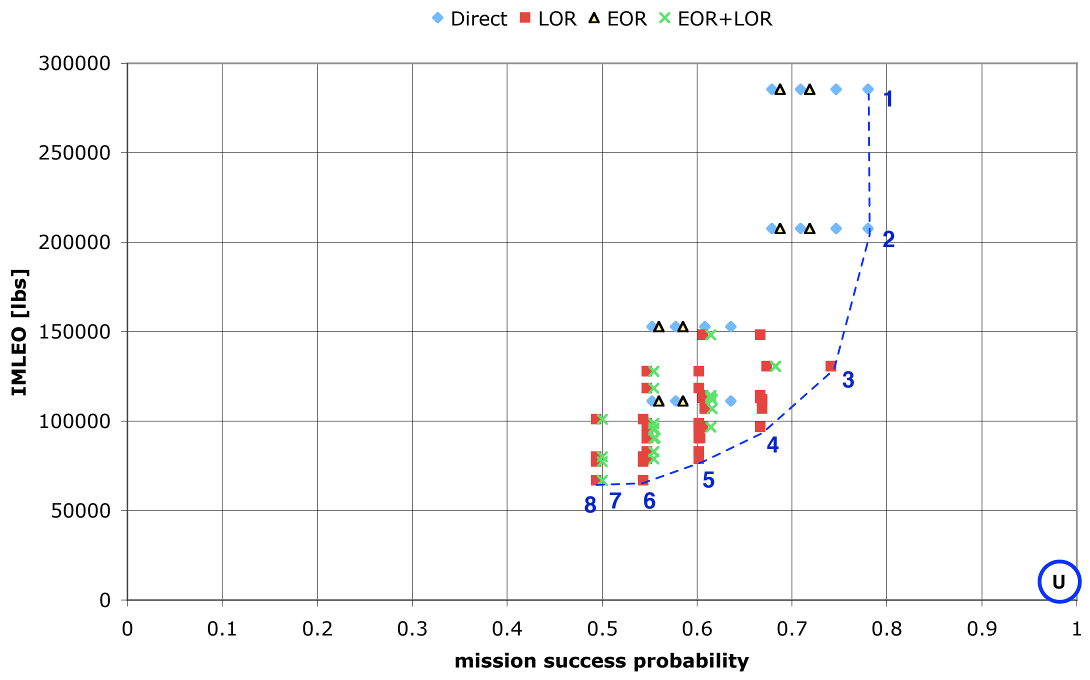


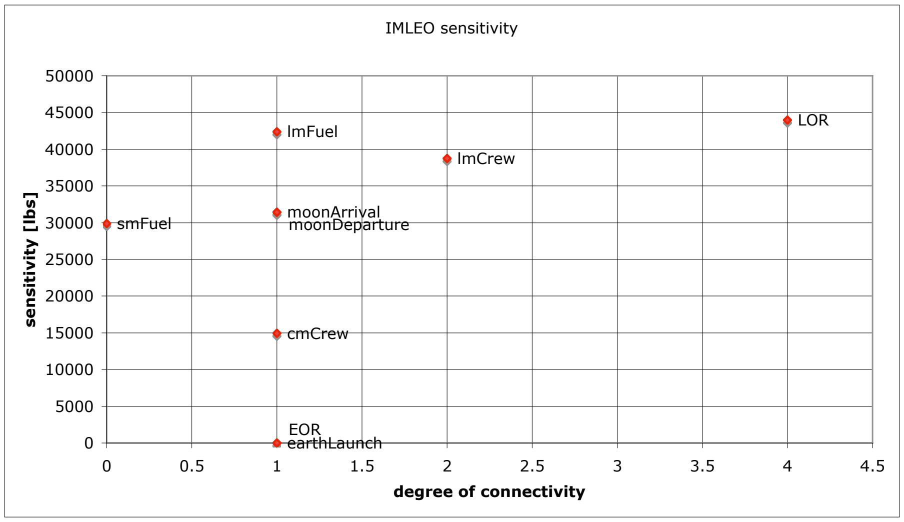
Figure 4-8: Apollo Pareto front plot comparing IMLEO to probability of mission success. Each point in the plot indicates a logically feasible combination of decision variable assignments. The blue dashed line indicates the Pareto front. The U indicates the utopia point. Note that the IMLEO numbers are consistent with contemporary information available in the early 1960's [Hou61a]. The actual IMLEO for the Apollo mission was about twice value the indicated by point 3.

4.5.2 Decision Space Views

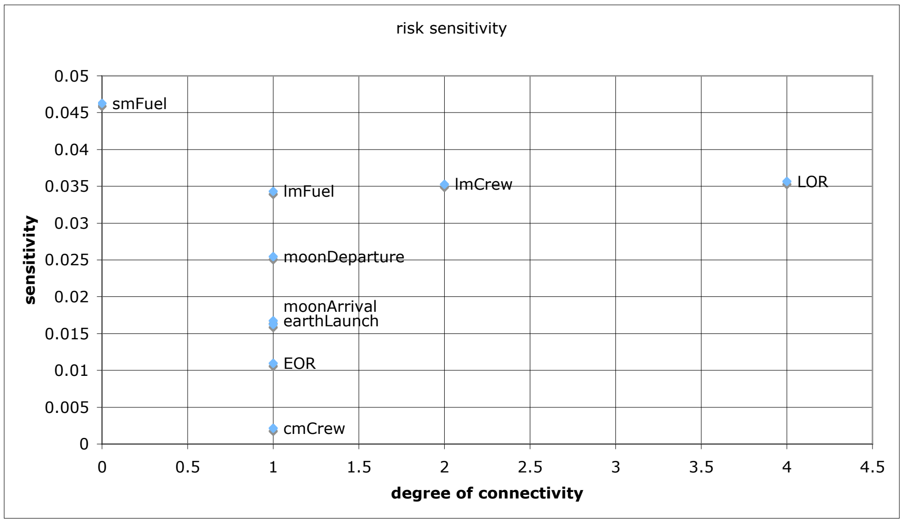
Section 3.6.1 introduced the concept of the decision space view. It is a plot of the decision variables on two axes which measure their potential impact on the architecture in different ways. The horizontal axis is the degree of connectivity, which measures the impact of a change in a decision variable on the set of alternatives for other decisions. The vertical axis is the Property Variable Sensitivity (PVS), which is a measure of the potential impact of a change in a decision variable on a system property.

The Apollo ADG contains two property variables: IMLEO and risk. The decision space view corresponding to the IMLEO property is shown in Figure 4-9a and the decision space view corresponding to the risk property is shown in 4-9b.

The most striking feature of the two decision space view plots is the location of the LOR decision variable in both plots. A conclusion that can be drawn from



(a) IMLEO



(b) risk

Figure 4-9: Decision Space Views for the IMLEO and risk properties. The plots show the sensitivity of the property functions to a change in a decision variable assignment versus the decision variable's degree of connectivity.

Figure 4-9 is that the LOR decision variable is a high impact decision both in terms of impact on system properties as well the degree of connectivity to other decisions. This conclusion concurs with historical records[Sea05]. A second conclusion that can be drawn from these two plots is that two other important decisions are the lunar module crew size (lmCrew) and lunar module fuel type (lmFuel). These decisions were also identified by Houbolt as critical factors that impact IMLEO and risk in his 1961 report on lunar orbit rendezvous for the Apollo mission[Hou61a, Kin05].

Referring to the decision space view breakdown from Figure 3-12, the decision variables can be categorized into the four categories. A decision category chart is shown in Table 4-7. According to the discussion in Section 3.6.1, the suggested order in which to address these decisions can be derived from the chart. The decision category chart highlights LOR and lmCrew as sensitive and strongly connected decisions in terms of both property variables. These two decision should be considered a high priority.

Table 4-7: Decision category chart for the Apollo ADG.

Category	IMLEO	risk
I: sensitive & strongly connected	LOR, lmCrew	LOR, lmCrew
II: sensitive, but weakly connected	lmFuel, moonArrival, moonDeparture, smFuel	lmFuel, smFuel, moonDeparture
III: insensitive & strongly connected		
IV: insensitive and weakly connected	cmCrew, EOR, earthLaunch	moonArrival, earthLaunch, EOR, cmCrew

The next group of decisions to be addressed includes lmFuel, moonArrival, moonDeparature, and smFuel. Note that by addressing the LOR decision before addressing this second group may greatly reduce the size of the remaining configuration space. If LOR is set to "yes", the moonArrival and moonDeparture decision variables can only be set to "orbit".[3]

The last group of decisions to be addressed consists of cmCrew, EOR, and earthLaunch. Note that the moonArrival decision variable was already addressed in the second group. The placement of the cmCrew decision variable in this last group also concurs with historical records. According to an interview with Robert Seamans [SKK05], an interview with John Houbolt [Kin05], and the lack of mention of formal analysis in historical records [MC04], the command module crew size was chosen almost arbitrarily. In the interview, Houbolt stated,

> *"In one of our informal, roundtable sessions, it was discussed that there had to be at least two. You needed a pilot in the orbiter, and you needed someone in the lander. With an airplane, there is usually a pilot, a co-pilot, and a navigator; so, arbitrarily, we picked a crew of three."*

Officially, the Apollo crew size of three astronauts was set as a ground rule for the 1961 Golovin committee to study launch vehicles[MC04]. According to some historical

[3]This assertion can be verified using the logical constraints in Section 4.2.4.

accounts, the ground rule was accepted because it was assumed that the astronauts would need to run Navy-like round-the-clock 8-hour shifts, which would require at least three astronauts [MC04, BGS79, Kin05]. Although John Houbolt suggested two crew alternatives for the lunar mission in his report on lunar orbit rendezvous [Hou61a], the crew size decision was never changed. This historical record also concurs with results from this study which determined that the command module crew size is not as impactful as the other decision variables.

4.6 Analysis of the Methodology

4.6.1 Impact of the Structural Reasoning Algorithms on Simulation Performance

The performance of the sorting algorithms is measured by their impact on the efficiency of the simulation algorithm. Since the Apollo problem has more decisions than the example problem in Chapter 3, it provides an opportunity to measure the efficiency impact of the ADGsort1 and ADGsort2 algorithms on a larger problem. The total number of possible orderings for the nine decisions is $9! = 362880$. Since this number is relatively large, a complete study of all possible orderings, like the one in Section 3.7.2, is not practical.[4] Instead, we use a random generator of variables orderings to generate a sample set of over 300 variable orders in addition to the ones produced by ADGsort1 and ADGsort2.

Figure 4-10 is a twin vertical axis plot of the performance of the simulation algorithm due to the orderings the two orderings produced by ADGsort1 and ADGsort2 as well as 330 randomly generated decision variable orderings. The left axis refers to the red points which indicate the total time of computation measured in seconds. The right axis refers to the blue diamond shaped points. This test was run on a PC running Windows 2003 sever equipped with an Intel Core 2 Duo 6600 Processor (2.40GHz) and 4 GB of RAM. The Java JVM was version 1.6.

The data in Figure 4-10 shows some similar features to the data from Figure 3-14. Both figures show a correlation between the number of tokens generated and the total computational time. For the Apollo ADG, the variable order produced by ADGsort1 performs better than the variable order produced by ADGsort2. However, compared

[4]Post-hoc analysis estimates that a complete study of all decision variable orderings would take $362880 * 7\text{sec}/60/60/24 \approx 30\text{days}$ to complete.

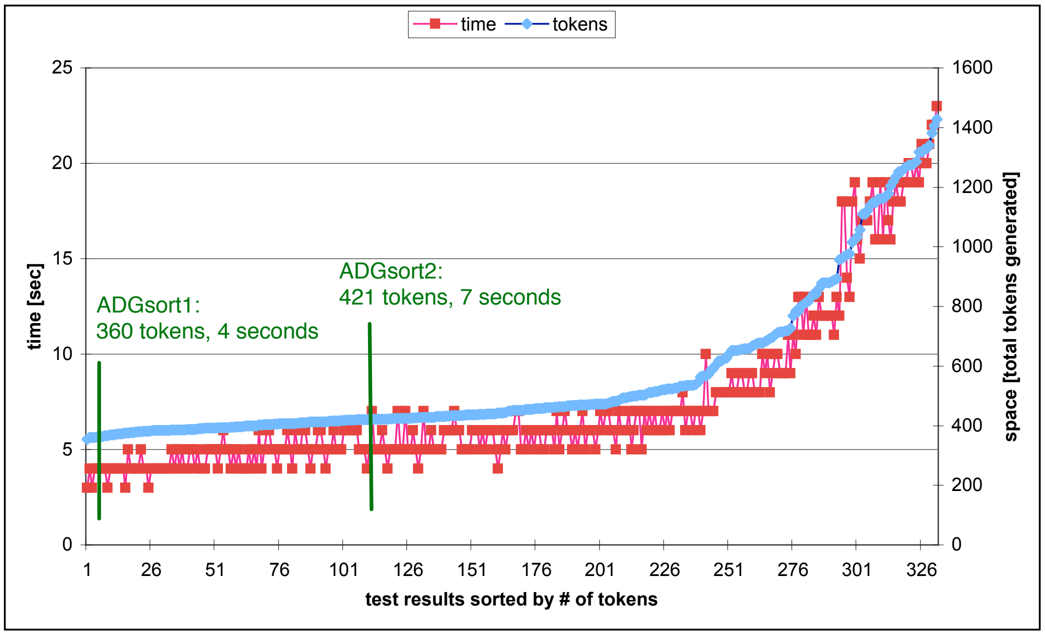


Figure 4-10: OPNBuild1 Performance for the Apollo Problem. This test of simulation performance includes the orderings produced by ADGsort1 and ADGsort2 as well as 330 random orderings. This is a 2-axis vertical plot, time [sec] and space [total tokens generated]. The performance of the orderings produced by ADGsort1 and ADGsort2 are indicated with the green vertical lines.

to the entire sample set of variable orderings, both of these variable orders produce OPN models which performed within 10% of the best ordering in the sample set. Additional studies of computational performance are presented in Section 5.7.1.

4.6.2 Comparison to Koo's study

In support of his thesis, "A Meta-language for Systems Architecting" [Koo05], B. Koo completed a similar retrospective study of early systems architecture decisions for Apollo. Koo's study modeled the Apollo system using Object-Process Network as a meta-language to build a domain specific language which represents that space of possible mission modes. The OPN model which corresponds to this approach is shown in Figure 4-11.

Koo's study used a motion language abstraction [Fra01] to model transitions and steady states of a mission modes as processes and objects in OPN. The conclusions of his study found that OPN's primitives were well-suited for constructing a discretized, finite abstraction of the continuous, infinite possible space of mission-modes. The results of the study identified nine categories of mission-modes, which included the

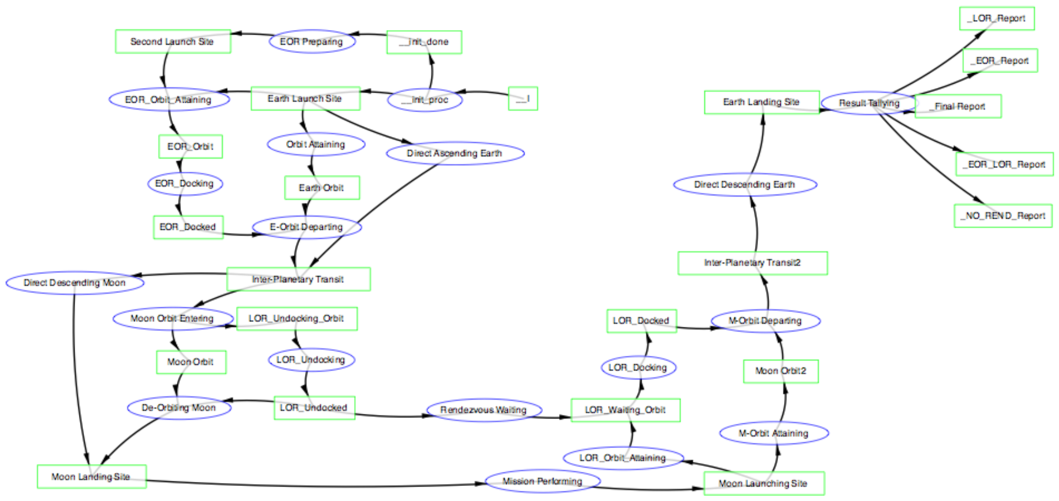


Figure 4-11: B. Koo's OPN model of a domain specific language representing the space of Apollo mission modes. (Source: [Koo05])

mission modes under consideration by NASA in the 1960's.

Koo's motion language approach to studying this architecture space is distinctly different from the ADG-based approach presented in this chapter. Koo's model contains an explicit sequence of mission operations, which are modeled as a discrete actions in a motion language. The notion of decision variables is implicitly captured by the sequence of branches in the network of objects and processes. Branches indicate alternative "trajectories" through the sequence of maneuvers.

Alternatives in Koo's model are evaluated in the order they would occur in the lunar mission. For example, the evaluation of the set of alternatives modeling the Earth departure type (to orbit or directly injected towards the moon) occurs before the evaluation of the alternatives for the lunar arrival type (to orbit or direct to surface). In contrast, in the ADG representation, decision variables are treated as an un-sequenced set of decisions. Alternatives are evaluated in an order that is computationally convenient, instead of explicitly evaluated in the order they would occur during the mission. The evaluation of the decision about the lunar arrival type could occur before or after the decision about the Earth departure type.

There are certain advantages to Koo's approach to the Apollo architecting problem. In the case of the Apollo mission mode problem, using a sequenced abstraction was appropriate since both the constraints between alternative maneuvers and the calculation of metrics are sequence dependent. Calculation of the rocket equation (Equation 4.1) depends on the sequence of maneuvers. In Koo's model the sequence

Table 4-8: Comparison of two approaches for modeling the decisions for the Apollo program.

Type of Decision	Koo's model	ADG model
Decisions about the sequence of maneuvers	EXPLICITLY represented by the sequence of objects and processes in OPN model recursively applied property functions.	IMPLICITLY represented through decision variables and property functions that determine implied sequence.
Decisions about static configuration of the system	IMPLICITLY represented in the OPN model as branches from objects to processes which change the state of the model.	EXPLICITLY represented as decision variables with a set of alternatives.

of maneuvers is explicit. In the ADG model, certain assumptions regarding the implied sequence of the maneuvers resulting for certain combinations of decisions are embedded in the IMLEO property function. These assumptions about maneuver order were required to properly calculate rocket performance. In Koo's approach, it's simple to add new types of mission mode maneuvers to the model. In the ADG approach, changing the set of maneuvers requires revisiting the IMLEO and risk property function calculation methods in order to verify if the assumptions about the sequence still hold.

There are also certain advantages to the ADG approach for modeling the Apollo decision problem. New decisions or alternatives about the static configuration of the system can easily be added by inserting a new decision variable or modifying an existing decision variable's set of alternatives. A new executable model, which takes into account the new information, can be generated automatically through running the structural reasoning and simulation algorithms.

Table 4-8 outlines the difference between the two modeling approaches, in terms of how they model architectural decisions. From this study, I conclude that the ADG is better suited for architectural selection models and not as well suited for strongly sequenced decision models.

4.7 Chapter Conclusions

The primary objective of this chapter was to demonstrate that ADG can be applied to a realistic architecting problem. This chapter achieves this goal by a presenting a retrospective architectural study of the Apollo program by modeling it as a set of interrelated decisions. The second objective of this chapter was to provide guidance for transforming architectural problems into decision problems. Section 4.2.1 presents four guiding principles which can be applied to an architecture problem in order to transform it into an ADG.

In this retrospective study of Apollo, it was demonstrated that decisions that have been historically considered the most important ones were recovered through ADG's methodology. Specifically, the decision about Lunar Orbit Rendezvous (yes or no) is clearly identified as a decision which shows up as highly impactful in the decision space views for mass and risk presented in Figure 4-9. Historical records [Sea96, BGS79, Han95, MC04, EMB$^+$78, Han99] and an interview with Associate Administrator of NASA during the Apollo Program, Dr. Robert Seamans[SKK05], concur that the Lunar Orbit Rendezvous mission mode decision was the most critical and consequential decision for the entire Apollo project. ADG analysis also identified lander crew size and lander fuel type as the next most impactful decisions. This concurs with the 1961 Houbolt report[Hou61a], which asserts that the three decisions LOR (yes or no), lander fuel type, and lander crew size were the most impactful decisions for the 1960's lunar exploration architecture.

5 Lunar Outpost Architecture Study

5.1 Overview

This chapter presents a study of the architecture for NASA's lunar outpost program. The objective of this chapter is to generate and evaluate feasible lunar outpost architectures in order to identify the architectural decisions which may have a strong impact on the delivery of value to the space program's stakeholders. Identifying the most impactful decisions allows the system architect of the lunar outpost program to prioritize decision-making activities.

The architectural study in this chapter demonstrates the application of ADG to a different type of architecting problem than the one presented in Chapter 4. In the previous chapter, the ADG framework was applied to the Apollo architecture problem. The Apollo study was primarily focused on determining the operational architecture and how changes in the architecture may impact the probability of mission success and the total mission mass. In this lunar outpost architecture study, the emphasis is on decision variables related to the static technology and operational configuration of the lunar outpost architecture.

The outline of this chapter follows the outlines of Chapters 3 and 4. First, the lunar outpost architecture problem is introduced. The four guiding principles for transforming the architecture problem into a decision problem are applied to the problem in order to develop the ADG representation. The remaining parts of the ADG framework are applied: structural reasoning, simulating, and viewing are applied to the lunar outpost problem to generate a set of feasible lunar outpost architectures and identify the most impactful decision variables. The chapter concludes with an analysis of the computational performance of the methodology, a discussion of the engineering results, and a summary of the findings of this chapter.

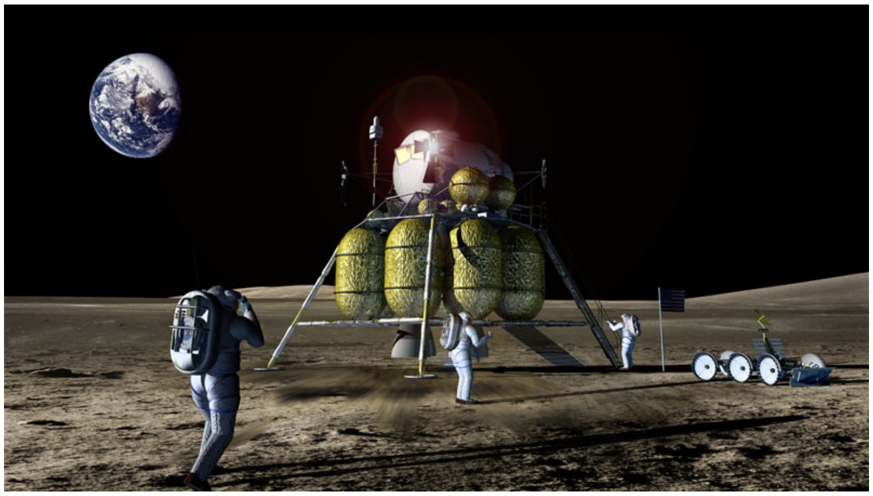


Figure 5-1: One of NASA's preliminary concepts for a new human lunar lander. (Source: [Exp05])

5.2 The Lunar Outpost Architecture Problem

In a speech called "The Vision for Space Exploration" by President Bush in January 2004 announced a new direction for NASA [Bus04]. It directed NASA to "build new ships to carry man forward into the universe, to gain a new foothold on the moon, and to prepare for new journeys to worlds beyond our own." The details of this vision were studied by the President's Commission for Moon, Mars, and Beyond [Ald04], which concluded that "The long-term, ambitious space agenda advanced by the President for robotic and human exploration will significantly help the United States protect its technological leadership, economic vitality, and security."

The NASA Authorization Act of 2005 [10905] codified the Vision for Space Exploration (VSE) by requiring NASA to develop a long-term lunar outpost:

> *"The Administrator shall establish a program to develop a sustained human presence on the Moon, including a robust precursor program to promote exploration, science, commerce, and U.S. preeminence in space, as a stepping stone to future exploration of Mars and other destinations."*

Several aspects of high-level requirements for the lunar exploration architecture can be derived from these short quotes: There shall be new ships (spacecraft), there shall be lunar exploration, there shall be a long-term presence on the moon, and there shall be preparation for Mars exploration. The high-level goals of the program can also be extracted from these statements: promote exploration, increase science knowledge, promote commerce, increase U.S. influence in the world, and develop of technology for future Mars exploration missions.

Although the proposed human lunar outpost is complex, unprecedented and novel, the challenges involved in the effort to architect a lunar outpost are distinctly different than the ones faced by Apollo's architects in the early 1960's. The main challenge for Apollo project was to identify an architecture that would successfully complete the technical task of landing a man on the moon and returning him safely to Earth by the end of the decade. The cost of the mission, science return, and other considerations were secondary[1]. In the case of the lunar outpost program, NASA is building upon more than forty years of experience in spaceflight. The program starts with the assumption that it is technically feasible to establish the lunar outpost. Controlling costs, controlling programatic risk, and ensuring consistent value delivery to stakeholders are the current priorities [TBB$^+$05, Dra05, Cam07, WH03].

The specific objective of this chapter is to identify the most potentially impactful decision variables in the lunar outpost architecture in order to support the development of an architecture which has a balanced approach to cost, risk, and value delivery. The results of this chapter can be used to prioritize the architecting process. The potential impact of some decisions must be identified early due to time-line constraints. For example, if NASA would like to develop a deployable semi-autonomous nuclear reactor as an outpost power source, it should make this decision as early as possible, since the development of a reactor requires a long lead time in terms of both technical development and ensuring long-term political support [BCE$^+$04].

NASA commissioned several studies to explore the technological challenges and begin to set bounds on the architectural space. In particular, the ESAS study [Exp05] and the LAT study[NAS06d] provide useful insights into the types of decisions NASA is currently considering in it's architecting process. In particular, "The Lunar Architecture Team Update", Reference [NAS07], NASA provided a list of six categories of decisions which it would like to address. These categories were: Transportation Vehicles, Habitation, Rover, EVA Systems, Surface Power, and Communication.

In our study we used a slightly modified list of six categories. Instead of two separate categories for Rover and EVA systems, the decisions relating to these issues have been combined into one category: Surface Mobility. In addition, we have added

[1]At its peak, the Apollo program consumed 5% of the annual U.S. federal budget [Orl04]. Controlling spending was not a high priority for the Apollo project [Sea05]. Detailed planning for lunar science activities was minimal until the mid to late 1960's [MC04].

a new category called Campaign Strategy, which includes high-level decisions, such as outpost locations. Table 5-1 lists the six categories. The two letter symbols next to each name is used as abbreviations for the categories throughout this chapter.

Table 5-1: Categories of lunar outpost decisions.

category	**description**
Campaign Strategy (CS)	Decisions involving outpost placement and campaign schedules.
Transportation Architecture (TA)	Decisions involving the configuration of the crew and equipment ascent from and descent to the lunar surface.
Surface Mobility (SM)	Decisions involving mobile landers and surface rovers.
Outpost power (OP)	Decisions involving how will power be generated and how energy will be stored.
Human Habitation (HH)	Decisions involving the human habitation requirements.
Communications (CM)	Decisions involving communications infrastructure and requirements.

The next section explains how this architectural problem can be represented as a decision problem through the use of ADG. By using ADG we gain some useful insight into the problem. For example, we can determine which architectural decision has have most potential impact and prioritize how they should be addressed.

5.3 Representing

Referring to the ADG cycle in Figure 3-1, this section transforms the available knowledge about the lunar outpost architecture candidate space into an ADG representation of the problem. This is achieved in the next four subsections by applying the four guiding principles for transforming an architecture problem into a decision problem. The details of the four principles were presented in Section 4.2.1.

5.3.1 Setting the Boundaries

The first guiding principle is to set of the boundaries of the architectural space under consideration. In the previous section, we pointed out that certain bounds on the lunar outpost architectural space have already been set by policy statements and congressional mandates. Specifically, in this study we assume that the form of the lunar exploration program always includes a lunar lander system similar to the one in Figure 5-1 and at least one long-term lunar outpost. The technical configuration of the crew transfer vehicle is not a subject of this study since this part of the program is already under development [NAS06c].

The “Lunar Architecture Update” document also provides a list of “options” under consideration by NASA for the lunar exploration program. The term “option” is used by NASA to describe unresolved technical decisions in the initial studies of lunar outpost architecture [NAS07]. The bounds of the decision study should include these decision variables. Specifically, the questions described in the Lunar Architecture Update include:

- Should all elements of the lunar outpost be delivered with the crew?
- Should some elements be delivered by un-crewed landers?
- Should the outpost habitat be a single, large pre-integrated habitat or be assembled from smaller modules?
- Should the lander be mobile? (Should it have the capability to move around the lunar surface?)
- Should the architecture include long-range pressurized rovers?
- Should nuclear power be used as a power source for the lunar outpost the surface?

Since these six questions are under active consideration by NASA, they should be captured within the bounds of the lunar outpost decision study.

We also include the idea of an intermediate outpost [HSWC07] within the bounds of this study. The intermediate outpost is a proposal for a pre-deployed, integrated, medium-duration (6 to 25 weeks) lunar outpost. It would be delivered on a single un-crewed cargo lander. The goal of the intermediate outpost is to provide a low-intial investment option for gaining early exploration experience. Decisions about whether to include or exclude the intermediate outpost and some questions about its location and energy are considered within the bounds of this study.

An additional bound on the study is that the decision variables included in this study are limited to high-level technical and policy questions about whether to include or exclude certain elements or features in the lunar exploration campaign. The study does not include decision variables about the details of individual instances of lunar exploration campaigns, such as specific deployment schedules, transportation logistics, or specific numbers, sizes, configurations of the elements. This bound was set because the lunar outpost architecture is still in the early planning stages and the objectives for the program are not nearly as specific as they were for Apollo. The objective of this study is to measure the potential impact of changes in decision variables on the the set of available alternatives for other decisions as well as the potential impact on system properties. By providing measures of the degree to which these decision variables impact the system, the decision-making process can be prioritized by the system architects.

5.3.2 Identify the Property Functions of Interest

The second guiding principle states that the property functions of interest should be identified before developing the set of decisions. By identifying what types of properties we would like to calculate first, we can insure that that the decision variables are inputs to these property functions. In Section 5.2 we indentified the architecting goals of the lunar exploration program: controlling cost, controlling risk and delivering value to the exploration program's stakeholders.

To measure the value delivery to stakeholders of the exploration program, we were able to leverage the public documentation of NASA's lunar exploration strategy. In these documents, NASA describes six "guiding themes" for the lunar exploration strategy[NAS06b, Tea07, CN06]: Human Civilization, Scientific Knowledge, Exploration Preparation, Global Partnerships, Economic Expansion, and Public Engagement. The themes, as defined by NASA, are presented in Figure 5-2.

The six themes are derived from the policy statements mentioned in Section 5.2 and are considered to be high-level objectives for the exploration program. Following the second principle, we chose to calculate property functions that measure the potential impact of the lunar outpost architecture to six property functions that are aligned with each of the six guiding themes.

In addition to these six properties, there is a need to calculate an architecture's impact on programatic costs and risk. After several iterations building the ADG

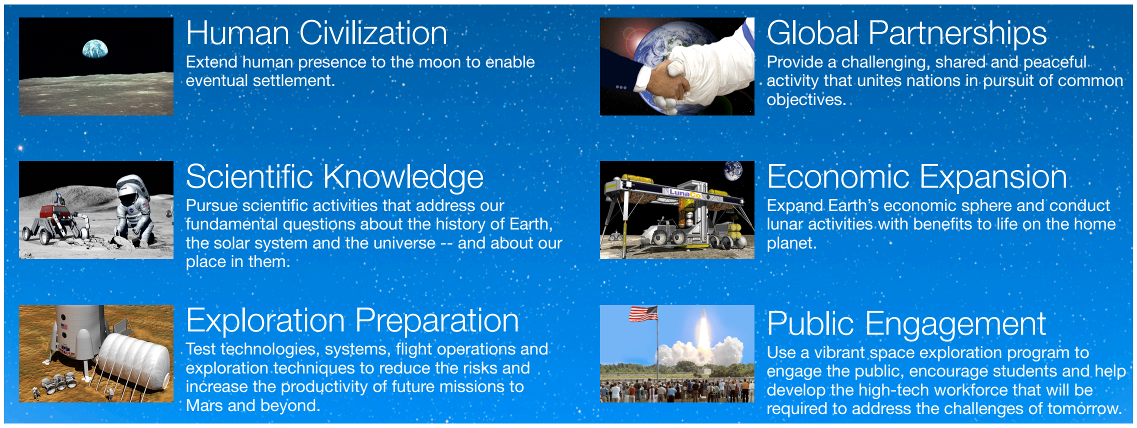

Human Civilization
Extend human presence to the moon to enable eventual settlement.
Global Partnerships
Provide a challenging, shared and peaceful activity that unites nations in pursuit of common objectives.
Scientific Knowledge
Pursue scientific activities that address our fundamental questions about the history of Earth, the solar system and the universe -- and about our place in them.
Economic Expansion
Expand Earth's economic sphere and conduct lunar activities with benefits to life on the home planet.
Exploration Preparation
Test technologies, systems, flight operations and exploration techniques to reduce the risks and increase the productivity of future missions to Mars and beyond.
Public Engagement
Use a vibrant space exploration program to engage the public, encourage students and help develop the high-tech workforce that will be required to address the challenges of tomorrow.

Figure 5-2: The six "guiding themes" for NASA's Lunar Exploration program. (Source: [NAS06b].)

model for the lunar outpost architecture study, it became evident that it would be difficult to calculate a high-resolution metric for both cost or risk, since the study deals with high-level questions like the ones listed in Section 5.3.1. Specific metrics for cost and risk require knowledge of specific implementation of the architecture. As a proximate metric[2], we calculate the relative impact of technology alternatives on programmatic development risk. The baseline assumption for this metric is that an architecture that includes many difficult-to-implement technologies will be both more costly and risky than an architecture that includes fewer difficult-to-implement technologies.

The seven property functions are described in detail in Section 5.3.7.

5.3.3 Capture The Architecturally Distinguishing Decisions

The third principle states that we should limit the set of decision variables to the architecturally distinguishing ones. An architecturally distinguishing decision is one that potentially changes the architecture's high-level concept, the mapping of function to forms, of the architecture. Therefore, they can be used to define decision variables that change the mapping of architectural functions to architectural forms.

In Reference [HWC07], Hofstetter, Wooster, and Crawley developed a modeling abstraction called "campaign elements" which can be used to create high-level models of lunar exploration campaigns which are under consideration by NASA. The abstrac-

[2]A proxy metric is a substitute measurement for something that is difficult to quantify in a direct manner[RCL+05, LCCR06]

tion partitions the architecture space by defining six types of elements that can be included or excluded from an exploration campaign.

The campaign elements, which include two types of sortie missions, two types of intermediate outposts, and two types of long-term outposts can be considered the major building blocks of a lunar exploration program. In this decision space study, we use their concept of campaign elements as a way to parameterize the lunar exploration system. The definition of each campaign element can be derived from the hierarchy of characteristics shown in Figure 5-3.

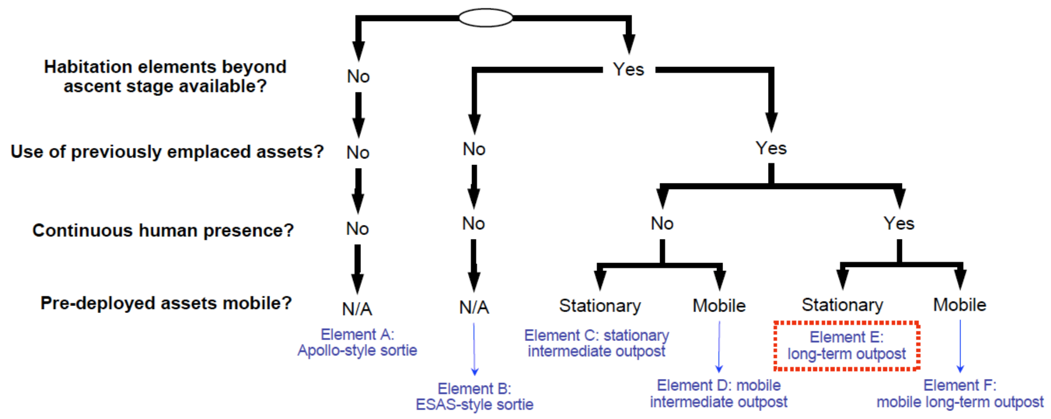

Habitation elements beyond ascent stage available?
Use of previously emplaced assets?
Continuous human presence?
Pre-deployed assets mobile?
No
Yes
No
No
Yes
No
No
No
Yes
N/A
N/A
Stationary
Mobile
Stationary
Mobile
Element A: Apollo-style sortie
Element B: ESAS-style sortie
Element C: stationary intermediate outpost
Element D: mobile intermediate outpost
Element E: long-term outpost
Element F: mobile long-term outpost

Figure 5-3: Derivation of campaign elements. (Source: [HWC07])

The six different campaign elements, A though F, are summarized in the follow list. For additional details see Reference [HWC07].

Element A: Apollo-style sortie. Element A is a basic mission to the moon which includes a lander with an ascent and decent stage. The habitation module is part of the ascent stage. In this study, we assume that Element A always exists in the lunar exploration system since it would also be used as crew transfer vehicle for the intermediate and long-term outposts. In line with current NASA plans, we assume a baseline design of a lander which is larger than Apollo's lander. It transports four crew as opposed to the two-crew Apollo Lunar Excursion Module(LEM).

Element B: ESAS-style long sortie. Element B is an extended duration and capability version of Element A. The long sortie is characteristically different from Element A because it provides for longer duration missions through additional habitable volume and/or an airlock module that may be left on the surface after the crew leaves the lunar surface[Exp05]. [3]

Element C: Stationary Intermediate Outpost. The intermediate lunar outpost concept was explained in Reference [HSWC07]. An intermediate is a pre-deployed integrated outpost which includes habitation, power generation, energy storage, and possibly surface mobility equipment. The basic concept of an intermediate outpost is that it would be delivered in one flight and crews would visit it using Element A for durations of 6 to 25 weeks, depending on location.

Element D: Mobile Intermediate Outpost. The mobile outpost concept is a variant of Element C. It is the same basic concept as element C, however it has the ability to move to different locations on the surface in between crew visits.

Element E: Stationary Long-term Outpost. Element E is a long-term outpost which has the goal of long-term continuous habitation. Element E may or may not be delivered in an integrated form in the way that the intermediate outpost would be. As a ground rule for this study we assume that all lunar outpost architectures include Element E.

Element F: Mobile Long-term Outpost. Element F is a long-term outpost similar to Element E, but has the ability to move around the surface to aid in exploration of additional sites. However, as we point out in Section 5.3.4, Element F is highly unlikely to be used since it requires transporting the outpost's habitable space, any associated ascent vehicles as well as power generation and energy storage equipment while a crew is in residence.

A specific lunar exploration campaign can be designed by specifying different numbers and sequences of the campaign elements. A campaign can be made up of a mixture of Apollo-style sorties (type A), ESAS-style long sorties (type B), stationary (type C) or mobile (type E) intermediate outposts, and a long term stationary (type E) or mobile outpost (type F). Figure 5-4 show examples of three campaigns built out of these elements. In this study, we assume that the lunar exploration program would include either stationary or mobile intermediate outposts and either stationary or mobile long-term outposts.

[3] The ESAS-style sortie is sometimes called the "Super Sortie"[NAS07].

	Flight number																			
	Year 1		Year 2		Year 3		Year 4		Year 5		Year 6		Year 7		Year 8		Year 9		Year 10	
	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B
Focus on long-term polar outpost	Test	A	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
Sortie, stationary intermediate and long-term outposts (both polar)	Test	B	C	C	C	C	C	E	E	E	E	E	E	E	E	E	E	E	E	E
Sortie and stationary intermediate outposts (polar locations)	Test	B	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C

Figure 5-4: Three example campaigns built from campaign elements. The overall goal of each campaign is listed in the left column. The sequence for deploying the campaign elements is listed in each row. (source: [HWC07])

5.3.4 Keep it Simple

The fourth principle from Section 4.2.1 states that the set of decisions should be kept as small as possible, but not too small. Specifically, the goal is to exclude decision variables that are not inputs to property functions or architecturally distinguishing. Through the use of the abstraction of campaign elements, the large space of possible campaigns is reduces to a small, finite set of building blocks which can be assembled together to form instances of campaigns. This compression through the use of the campaign element abstraction keeps the decision space as small as possible.

As pointed out in the section about the boundaries of the study (Section 5.3.1), the study doesn't include decisions about the transportation of the campaign elements or supply logistics. The objective is to address the high-level questions about which types of elements and technologies exist in the architecture.

The following four modeling assumptions were used in this study to simplify the set of decision variables:

1. We assume that a long-term outpost campaign element (either E or F) is always included in the campaign. Although other lunar campaigns are possible which do not include a long-term outpost, the nature of this study about lunar outpost architectures implicitly require that a long-term outpost exists.

2. There is no long-term mobile outpost (element F). Campaign element F is seen as unlikely since it would require moving an outpost as well as a power system and other necessary support equipment around the surface of the moon with the crew. This operation is overly complex in the lunar environment.

3. We assume that campaign element A, the Apollo-style sortie, always exists in the architecture since this element would be used at least once as a test flight

and nominally as the crew transfer vehicle to and from long-term or intermediate outposts.

4. We assume that the architecture may include either element C, or element D, or neither element C or D. It may not include both element C and element D. This is based on the modeling assumption that a mobile intermediate outpost (element D) subsumes the functionality of a stationary intermediate outpost (element C).

5.3.5 Lunar Outpost Decision Variables

This subsection lists the decision variables used in this study. They were derived after several iterations of applying the four guiding principles for transforming the architecture problem into a decision problem (described above). Table 5-2 is a list of the decision variables for the lunar outpost in the form of a morphological matrix (see Section 2.3.1). Following the table, is a list where each decision is described in detail.

The set of decision variables includes variables from each of the six high-level categories of decisions listed in Table 5-1. The decision variables used in this study are primarily focused on determining which elements are included or excluded, their high-level configuration, and allowed lunar locations of the elements in the exploration campaign. In addition, two policy decisions are included: Which types of nuclear power systems will be allowed and whether or not anytime return (see [Exp05]) is required for the intermediate and/or long-term outpost.

CS:longSortie : include long sortie B, {no, yes}. This decision determines whether or not campaign element B (ESAS-style extended sortie) is included in the campaign.

CS:intOutpost : include intermediate outpost CD, {no, yes}. This decision determines whether or not one of the intermediate outpost elements, B or C, are included in the campaign.

CS:intOutpostLoc : intermediate outpost locations allowed, {NA, polarOnly, eqAndPolar, globalAccess}. This decision determines where intermediate outposts are permitted. The alternative NA is used when this decision is not relevant.

Table 5-2: Morphological Matrix of the Outpost Decisions.

cat	shortID	Decision	units	alt A	alt B	alt C	alt D
CS	longSortie	include long sortie B	none	no	yes		
CS	intOutpost	include int outpost CD	none	no	yes		
CS	intOutpostLoc	int outpost locations allowed	none	NA	polarOnly	eqAndPolar	globalAccess
CS	outpostLoc	outpost locations allowed	none	polarOnly	eqAndPolar	globalAccess	
TA	anytimeOutpost	anytime return for outpost?	none	no	yes		
TA	anytimeIntO	anytime return for int outpost?	none	no	yes		
SM	pressRov	pressurized rover	none	no	yes		
SM	mobileOutpost	include mobile outpost D	none	no	yes		
OP	outpostEnergyStorage	outpost energy storage	none	none	batteries	fuelcell	
OP	outpostPowerGen	outpost power generation	none	solar	DIPS	nucReactor	
OP	ioEnergyStorage	IntOenergy storage	none	NA	none	batteries	fuelcell
OP	ioPowerGen	IntOpower generation	none	NA	solar	DIPS	
OP	nuclearPolicy	nuclear policy	none	none	rtg_like	rtg_like_or_fission	
HH	habGround	ground level access for outpost	none	no	yes		
HH	habOffload	habitat offload for outpost	none	no	yes		
HH	sepAirlock	separate airlock required	none	no	yes		
HH	longSortieHab	long sortie habitat	none	NA	airlockMod	outpostHab	other
HH	habAssembly	lunar surface hab assembly	none	no	yes		
HH	pressConnections	type of pressurized connections	none	none	habRover	habHab	both
CM	commRepeater	need a repeater?	none	no	yes		
CM	dataRate	communications data rate	none	telemAndVideo	HDLive		
CM	LOItelemetryReq	is LOI telemetry required?	none	no	yes		

CS:outpostLoc : outpost locations allowed, {polarOnly, eqAndPolar, globalAccess}. This decision determines which locations are permitted for the long-term outpost.

TA:anytimeOutpost : Is anytime return required for the long-term outpost?, {no, yes}. An anytime return requirement states that the crew must be able to return to Earth within 5 days or less from any point on the lunar surface (see Chapter 4 of Reference [Exp05]). This requirement may limit where the long-term outpost is placed on the lunar surface.

TA:anytimeIntO : Is anytime return required for the intermediate outpost?, {no, yes}. An anytime return requirement states that the crew must be able to return to Earth within 5 days or less from any point on the lunar surface (see Chapter 4 of Reference [Exp05]). This decision may limit where an intermediate outpost may be placed on the lunar surface.

SM:pressRov : pressurized rover, {no, yes}. This decision specifies whether or not there is a pressurized rover available for surface exploration of the moon. Pressurized rovers enable astronauts to spend more time away from the outpost.

SM:mobileOutpost : include mobile outpost D, {no, yes}. This decision specifies whether or not a mobile outpost exists in the lunar campaign.

OP:outpostEnergyStorage : long-term outpost energy storage, {none, batteries, fuelcell}. This decision determines the type of energy storage technology for the long-term outpost. In some case the energy storage technology is insignificant because the outpost may be powered by a constant power source such as a nuclear reactor or is placed at lunar pole such that it is illuminated by sufficient sunlight. In this case, the alternative "none" is selected to indicate that the energy storage technology is not significant (it could be short-term batteries or open-loop fuel cells). The other two alternatives for energy storage are long-term batteries and regenerative fuel cells.

OP:outpostPowerGen : long-term outpost power generation, {solar, DIPS, nucReactor}. This decision determines if the outpost's power is primarily generated by photovoltaic solar cells, DIPS (Dynamic Isotope Power System [Sch07, Ker05]), or a fission reactor (nucReactor).

OP:ioEnergyStorage : intermediate outpost energy storage, {NA, none, batteries, fuelcell}. This decision has the same set of alternatives as the long-term outpost energy storage decisions with one addition. The alternative "NA" applies in the case that this decision variable is irrelevant in the case there the intermediate outpost is not included in the campaign.

OP:ioPowerGen : intermediate outpost power generation, {NA,solar, DIPS, nucReactor}. This decision has the same set of alternatives as the long-term outpost energy storage decision with one addition: The alternative "NA" applies in the case that this decision variable is irrelevant in the case there the intermediate outpost is not included in the campaign.

OP:nuclearPolicy : nuclear policy, {no, rtg_like, rtg_like_or_fission}. This policy decision indicates what types of nuclear technology is allowed. The option rtg_like indicates that an RTG (radioisotope thermoelectric generator) based power sources, like DIPS [Sch07, Ker05] are allowed. The option rtg_like_or_fission indicates that RTG, DIPS, or nuclear reactors are allowed.

HH:habGround : ground level access for long-term outpost required, {no, yes}. This decision indicates whether or not ground-level human access is required for the outpost habitat. This policy decision affects other decisions about offloading and airlock modules.

HH:habOffload : habitat offload for outpost, {no, yes}. This decision indicates whether or not the habitats for the long-term outpost will be offloaded from a lander or remain on top of a lander.

HH:sepAirlock : separate airlock required, {no, yes}. This decision specifies whether or not a separate airlock module is required. A separate airlock module may be necessary if (lunar) ground access is required and from outpost habitats are not offloaded from the lander.

HH:longSortieHab : long sortie habitat, {NA, airlockMod, outpostHab, other}. If there is no long sortie (element B), then this decision is set to "NA". If there is a long sortie, the habitat can be a modification of the airlock module (airlockMod), a modification of the long-term outpost habitat (outpostHab), or custom module (other).

HH:habAssembly : lunar surface habitat assembly for the long-term outpost, {no, yes}. This decision indicates whether or not assembly of separate habitats is required for the lunar surface habitat.

HH:pressConnections : type of pressurized connections, {none, habRover, habHab, both}. This decision indicates the types of pressurized connections available in the systems architecture. The alternative none indicates no pressurized connections. The alternative habRover indicates there are connections between a pressurized rover and habitat modules. The alternatives habHab indicates that there are pressurized connections between habitat modules. The alternative both indicates that there are both habRover and habHab connections.

CM:commRepeter : does the architecture include a communication repeater?, {no, yes}. This decision indicates whether or not the architecture includes a communications repeater in space to facilitate continuous communication from the distal half of the moon to mission control on Earth. This repeater could be placed in lunar orbit or at a Earth-moon Lagrange point [Far67].

CM:dataRate : communications data rate, {telemAndVideo, HDLive}. This decision specifies the bandwidth of the moon-Earth communication system. The alternative telemAndVideo indicates that there is a standard amount of transmission bandwidth available. The alternative HDLive indicates that the bandwidth is large enough to handle normal telemetry requirements as well as multiple live high-definition digital television signals.

CM:LOItelemetryReq : is LOI telemetry required?, {no, yes}. This decision indicates whether or not Lunar orbit insertion telemetry is required. LOI telemetry is operationally desirable since LOI burns occur on the far side of the moon.

5.3.6 Lunar Outpost Logical Constraints

The logical constraints for the outpost problem constrain the set of feasible combination of assignments to decision variables. Each constraint is listed in equational form in table 5-3. The format of the Table is the same as the separable property table for the example problem presented in Section 3.3.2. A explanation and justification for each constraint is given in the following list:

airlockNeeded : scope:habGround,habOffload,sepAirlock. This three-way constraint specifies that if ground access is required for the long-term outpost and the outpost habitats are not offloaded, then there is a implied requirement for a separate airlock module.

pressRoverConnection : scope:pressConnections,pressRov. This constraint says that a pressurized rover must exist for pressurized connections to be set to either (habRover) or (both).

habPressConnection : scope:pressConnections,habAssembly. This constraint requires that if the architecture includes habitat assembly, it must also include pressurized connections between habitats.

assemblyOffloadConstraint : scope:habAssembly,habOffload. This constraint requires that if the long-term outpost habitat is assembled, then it is implied that the outpost habitat must be offloaded from a lander.

longSortieHabCommon : scope:habAssembly,longSortieHab. This constraint relates the decision about the habitat assembly to the decision about the long sortie habitat. If the outpost habitat is assembled, this constraint allows the outpost habitat to be a common module with the long sortie habitat. However, if the outpost hab is not assembled, this implies that it is a large module, therefore the constraint does not allow the long sortie habitat to be common with the outpost habitat.

Table 5-3: Logical Constraints for the Lunar Outpost Problem.

name	scope	equation
airlockNeeded	habGround,habOffload,sepAirlock	((sepAirlock == yes && habGround == yes && habOffload == yes) || (sepAirlock == no && ((habGround == no && habOffload == no) || (habGround == yes && habOffload == yes))))
pressRoverConnection	pressConnections,pressRov	(pressRov == no && (pressConnections == none || pressConnections == habHab)) || (pressRov == yes)
habPressConnection	pressConnections,habAssembly	(habAssembly == yes && pressConnections != none) || (habAssembly == no)
assemblyOffloadConstraint	habAssembly,habOffload	(habOffload == no && habAssembly == no) || (habOffload == yes)
longSortieHabCommon	habAssembly,longSortieHab	((longSortieHab == outpostHab && habAssembly == no) == 0)
longSortieHabConstraint	longSortie,longSortieHab	(longSortie == no && longSortieHab == NA) || (longSortie == yes && longSortieHab != NA)
longSortieAirlockMod	sepAirlock,longSortieHab	(sepAirlock == yes && (longSortieHab == NA || longSortieHab == airlockMod)) || (sepAirlock == no && longSortieHab != airlockMod)
nucConstraint	nuclearPolicy,outpostPowerGen	(nuclearPolicy == none && outpostPowerGen == solar) || (nuclearPolicy == rtg_like && outpostPowerGen != nucReactor) || (nuclearPolicy == rtg_like_or_fission)
nucConstraintLO	nuclearPolicy,ioPowerGen	((nuclearPolicy == none && ioPowerGen == DIPS) ==0)
mobileOutpostLocation	mobileOutpost,intOutpostLoc	(mobileOutpost == yes && (intOutpostLoc == eqAndPolar || intOutpostLoc == globalAccess)) || (mobileOutpost == no)
outpostLocationEnergy	outpostEnergyStorage,outpostPowerGen, outpostLoc	(outpostPowerGen!=solar)||((outpostPowerGen==solar&&((outpostLoc == polarOnly)||(outpostLoc!=polarOnly&&outpostEnergyStorage==fuelcell))))
ioLocationEnergy	ioEnergyStorage,ioPowerGen, intOutpostLoc	((ioPowerGen == solar && (intOutpostLoc == eqAndPolar || intOutpostLoc == globalAccess) && ioEnergyStorage != fuelcell) == 0)
ioMobileConstraint	intOutpost,mobileOutpost	(intOutpost == yes) || (intOutpost == no && mobileOutpost == no)
IoLocConstraint	intOutpost,intOutpostLoc	(intOutpost == no && intOutpostLoc == NA) || (intOutpost == yes && intOutpostLoc != NA)
anytimeIntOconstraint	anytimeIntO,intOutpostLoc	((anytimeIntO == yes && (intOutpostLoc == globalAccess || intOutpostLoc == NA)) == 0)
anytimeOutConstraint	anytimeOutpost,outpostLoc	((anytimeOutpost == yes && outpostLoc == globalAccess) == 0)
ioEnergyConstraint	intOutpost,ioEnergyStorage	(intOutpost == no && ioEnergyStorage == NA) || (intOutpost == yes && ioEnergyStorage != NA)
ioPowerGenRequired	intOutpost,ioPowerGen	(intOutpost == no && ioPowerGen == NA) || (intOutpost == yes && ioPowerGen != NA)
LOItelemetry	LOItelemetryReq,commRepeater	(LOItelemetryReq == no && commRepeater == no) || (LOItelemetry == yes)

longSortieHabConstraint : scope:longSortie,longSortieHab. This constraint specifies that if there is no long sortie, then the decision about the long sortie habitat is not applicable (set to the NA alternative).

longSortieAirlockMod : scope:sepAirlock,longSortieHab. This constraint specifies that if there is a separate airlock module, then the set of feasible alternatives for the decision variable longSortieHab includes the alternative to modify the airlock to be used as a long sortie habitat. If there is no separate airlock module, then that alternative is not feasible.

nucConstraint : scope:nuclearPolicy,outpostPowerGen. This constraints links the nuclear policy to the available technologies for the long-term outpost's power generation. A nuclear policy of "none" restricts power generation on the lunar surface to solar power only. The alternative "rtg_like" enables either non-fissile nuclear power generation (DIPS) or solar power generation. The alternative "rtg_like_or_fission" enables all three types of power generation: nuclear reactor, DIPS, or solar.

nucConstraintIntO : scope:nuclearPolicy,ioPowerGen. This constraint links nuclear policy with the intermediate outpost's power generation technology. A nuclear policy of "none" restricts power generation on the lunar surface to solar power only. The alternative "rtg_like" enables either non-fissile nuclear power generation (DIPS) or solar power generation.

mobileOutpostLocation : scope:mobileOutpost,intOutpostLoc. This constraint links the decision of whether or not to have a mobile intermediate outpost with the locations for an intermediate outpost. If the mobile outpost exists, this constraint eliminates the possibility of "polarOnly". In this case, it would make more sense to use a stationary outpost and cover the polar circle with pressurized rovers. If the mobile intermediate outpost was moved beyond the polar circle, it would lose its anytime abort capability as well as need a different energy storage system. In this decision model, the possibility of having a mobile intermediate outpost placed at the poles is preserved by allowing the location option of "globalAccess". However, if global access is selected, the mobile intermediate outpost must also meet the power, energy storage, and abort requirements constrained by global access.

outpostLocationEnergy : scope: outpostEnergyStorage,outpostPowerGen,outpostLoc. This three-way constraint links the energy storage technology for the long-term outpost with its location and power generation technology. If the allowed long-term outpost location is global access, then batteries are not a feasible energy storage device for storing 14 days of energy (the lunar night) if solar power is the primary power source. If the outpost location is polar or if solar power is not used, then all energy storage options are available.

ioLocationEnergy : scope: ioEnergyStorage,ioPowerGen, intOutpostLoc. This three-way constraint links the energy storage technology for the intermediate outpost with its location and power generation technology. If the allowed intermediate outpost location is global access, then batteries are not a feasible energy storage device for storing 14 days of energy (the lunar night) if solar power is the primary power source. If the outpost location is polar or if solar power is not used, then all energy storage options are available.

ioMobileConstraint : scope:intOutpost,mobileOutpost. This constraint requires that the decision variable determining whether or not there is an intermediate outpost is set to "yes" in order for the variable determining whether or not there is a mobile intermediate outpost to be set to "yes".

ioLocConstraint : scope:intOutpost,intOutpostLoc. This constraint states that if there is no intermediate outpost, then the location of the intermediate outpost is constrained to the alternative "NA".

anytimeIntOconstraint : scope:anytimeIntO,intOutpostLoc. This constraint states that if anytime return is required for the intermediate outpost, then the alternative "globalAccess" for the intermediate outpost's location is not allowed. See the decision variable anytimeOutpost for the details of the anytime return requirement.

anytimeOutConstraint : scope:anytimeOutpost,outpostLoc. If anytime return is required for the long-term outpost, then the alternative "globalAccess" for outpost's location is not allowed. See the decision variable anytimeOutpost for the details of the anytime return requirement.

ioEnergyConstraint : scope:intOutpost,ioEnergyStorage. This constraint states that if there is no intermediate outpost (intOutpost=no), then the intermediate outpost energy storage technology decision variable is not applicable (ioEnergyStorage=NA).

ioPowerGenRequired : scope:intOutpost,ioPowerGen. This constraint states that if there is no intermediate outpost, then the power generation technology for the intermediate outpost is NA.

LOItelemetry : scope:LOItelemetryReq,commRepeater. This constraint states that if lunar orbit insertion telemetry is required, then the architecture must include an in-space communications repeater.

5.3.7 Lunar Outpost Property Variables and Property Functions

Section 5.3.2 identified seven system properties of interest. In this section, the details of these seven properties and their associated property functions are listed. Six of the property functions approximate how a particular decision contributes to each of NASA's six themes for the exploration program. In addition to these six beneficial measures of performance, the seventh metric measures how particular alternatives for a decision variable contributes to the development risk of the exploration program. The seven metrics are listed in Table 5-4.

Table 5-4: Lunar Outpost Property Functions.

shortName	Description	type
`devRiskFunc`	Development Risk	multiplicitive
`humanCivFunc`	Human Civilization benefit	additive
`scienceKnowledgeFunc`	Science Knowledge benefit	additive
`marsPrepFunc`	Mars Exploration Preparation benefit	additive
`globalPartnershipsFunc`	Global Partnerships benefit	additive
`econExpansionFunc`	Economic Expansion benefit	additive
`publicEngFunc`	Public Engagement benefit	additive

As explained in Section 5.3.2, the development risk property function is used as a proximate metric to measure the impact on programmatic cost and risk. The calculation of development risk (`devRiskFunc`) is done using a standard probabilistic risk assessment approach, similar to the ones presented in Section 3.3.2 for the example problem and Section 4.2.5 for the Apollo problem. Alternatives were given risk scores of 1.00 for no effect on risk, 0.99 a small increase in risk, 0.95 for a moderate increase in risk, and 0.92 for a large increase in risk. The development risk property function is a multiplicatively separable function (see Section 3.3.1). The values in for development risk listed in Tables 5-5, 5-6, and 5-7 were gathered through the a survey of aerospace experts.

The six other property functions were developed by mapping the data from a NASA study of two hundred specific lunar exploration objectives (see Reference [NAS06a]) onto additively separable property functions for each lunar exploration theme. These property functions quantify how strongly the selection of an alternative for a decision variable potentially benefits the lunar exploration program in one of the six themes. The entries in the property table for a decision variable's alternative can be either 0, 1, or 2. An entry of 0 indicates that a particular alternative for a decision does not positively effect that property. An entry of 1 indicates that this alternative is likely to positively effect that property. An entry of 2 indicates that the alternative has a strong positive effect on that property. Detailed descriptions of each property variable and the rules for how the benefit scores were assigned to each property function are listed below:

Human Civilization Property. Goal: "Extend human presence to the Moon to enable eventual settlement."

Rules: 0 = no direct effect on human civilization goals, 1 = directly provides technologies or experience benefitting human civilization goals, 2=provides extensive experience or provides critical technology development advancing human civilization goals.

Science Knowledge Property. Goal: "Pursue scientific activities that address fundamental questions about the history of Earth, the solar system and the universe – and about our place in them."

Rules: 0 = no direct benefit for science goals, 1 = directly provides technologies or opportunities to increase scientific data gathering, 2 = provides critical technologies or provides opportunities for extensive scientific data gathering.

Mars Exploration Preparation Property. Goal: "Test technologies, systems, flight operations and exploration techniques to reduce the risks and increase the productivity of future missions to Mars and beyond."

Rules: 0 = no direct impact on Mars exploration preparation goals, 1 = directly provides necessary exploration experience or technologies needed for Mars exploration, 2 = provides extensive experience or technologies which would strongly benefit Mars exploration preparation goals.

Global Partnerships Property. Goal: "Provide a challenging, shared and peaceful activity that unites nations in pursuit of common objectives."

Rules: 0 = provides no direct impact on global partnership goals, 1 = directly provides an opportunity for participation by international partners, 2 = strongly benefits global participation goals.

Economic Expansion Property. Goal: "Expand Earth's economic sphere, and conduct lunar activities with benefits to life on the home planet."

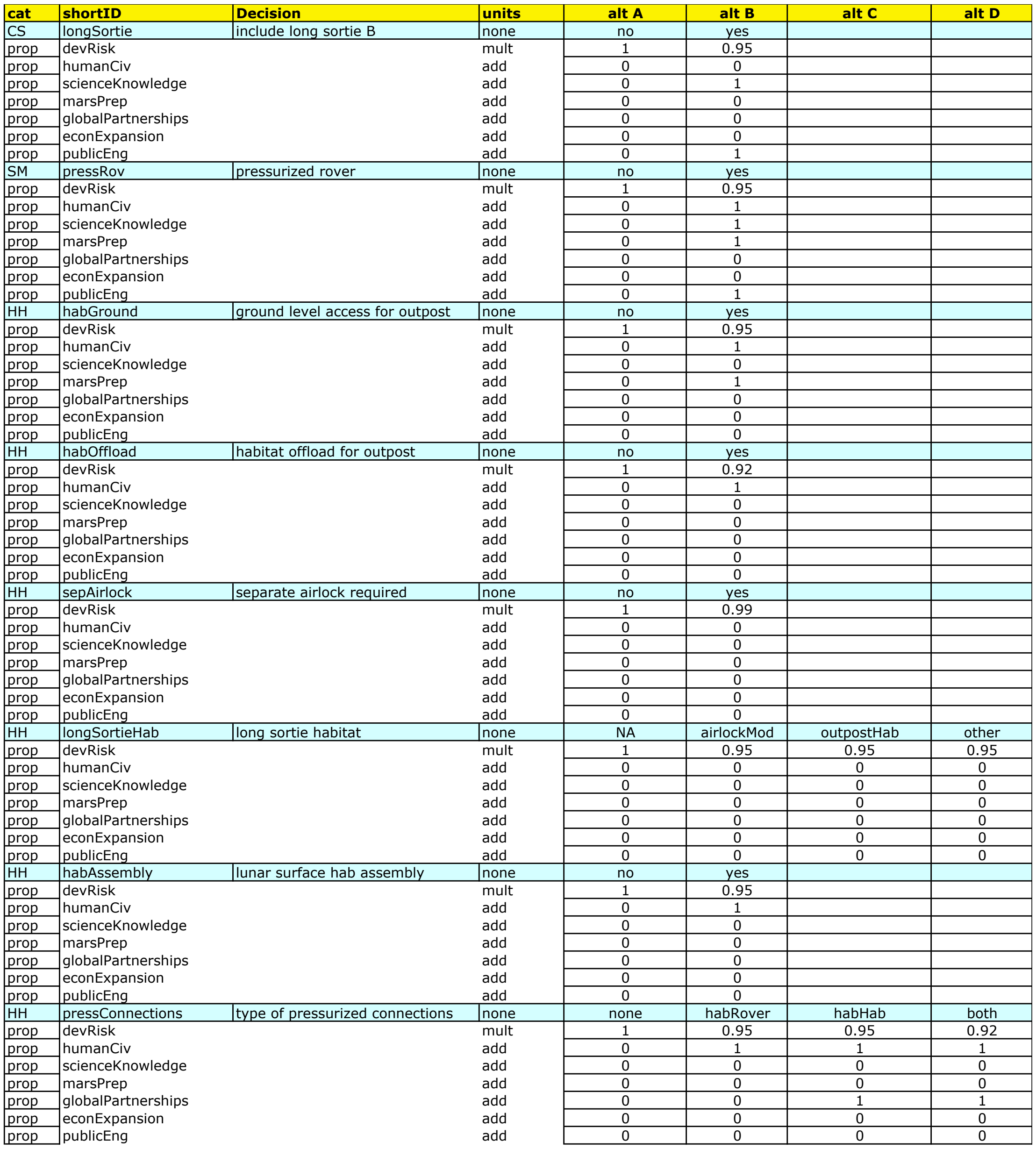
Rules: 0 = no direct impact on economic expansion goals, 1 = provides an opportunity or technology which may be beneficial to commercial stakeholders, 2 = provides a technology or opportunity which provides an especially strong benefit towards economic expansion due to lunar exploration.

Public Engagement Property. Goal: "Use a vibrant space exploration program to engage the public, encourage students and help develop the high-tech workforce that will be required to address the challenges of tomorrow."

Rules: 0 = no direct impact on public engagement, 1 = directly provides a means for increased public awareness or participation, 2 = strongly impacts public interest, awareness, or participation in the lunar exploration program.

The value of each decision variable's alternative's effect on the property functions are listed in Tables 5-5, 5-6, and 5-7.

Table 5-5: Lunar Outpost Separable Property Table 1/3. This table is continued in the next figure.



cat	shortID	Decision	units	alt A	alt B	alt C	alt D
CS	longSortie	include long sortie B	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	0		
prop	scienceKnowledge		add	0	1		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	1		
SM	pressRov	pressurized rover	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	1		
prop	scienceKnowledge		add	0	1		
prop	marsPrep		add	0	1		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	1		
HH	habGround	ground level access for outpost	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	1		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	1		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		
HH	habOffload	habitat offload for outpost	none	no	yes		
prop	devRisk		mult	1	0.92		
prop	humanCiv		add	0	1		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		
HH	sepAirlock	separate airlock required	none	no	yes		
prop	devRisk		mult	1	0.99		
prop	humanCiv		add	0	0		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		
HH	longSortieHab	long sortie habitat	none	NA	airlockMod	outpostHab	other
prop	devRisk		mult	1	0.95	0.95	0.95
prop	humanCiv		add	0	0	0	0
prop	scienceKnowledge		add	0	0	0	0
prop	marsPrep		add	0	0	0	0
prop	globalPartnerships		add	0	0	0	0
prop	econExpansion		add	0	0	0	0
prop	publicEng		add	0	0	0	0
HH	habAssembly	lunar surface hab assembly	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	1		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		
HH	pressConnections	type of pressurized connections	none	none	habRover	habHab	both
prop	devRisk		mult	1	0.95	0.95	0.92
prop	humanCiv		add	0	1	1	1
prop	scienceKnowledge		add	0	0	0	0
prop	marsPrep		add	0	0	0	0
prop	globalPartnerships		add	0	0	1	1
prop	econExpansion		add	0	0	0	0
prop	publicEng		add	0	0	0	0

Table 5-6: Lunar Outpost Separable Property Table 2/3. This table is continued on the next page.

cat	shortID	Decision	units	alt A	alt B	alt C	alt D
CS	intOutpost	include int outpost CD	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	1		
prop	scienceKnowledge		add	0	1		
prop	marsPrep		add	0	1		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	1		
SM	mobileOutpost	include mobile outpost D	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	1		
prop	scienceKnowledge		add	0	1		
prop	marsPrep		add	0	1		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	1		
prop	publicEng		add	0	1		
CS	intOutpostLoc	int outpost locations allowed	none	NA	polarOnly	eqAndPolar	globalAccess
prop	devRisk		mult	1	0.95	0.95	0.95
prop	humanCiv		add	0	1	1	2
prop	scienceKnowledge		add	0	1	1	2
prop	marsPrep		add	0	0	0	0
prop	globalPartnerships		add	0	0	0	0
prop	econExpansion		add	0	0	0	1
prop	publicEng		add	0	0	0	0
CS	outpostLoc	outpost locations allowed	none	polarOnly	eqAndPolar	globalAccess	
prop	devRisk		mult	1	0.95	0.92	
prop	humanCiv		add	1	1	2	
prop	scienceKnowledge		add	1	1	1	
prop	marsPrep		add	0	0	0	
prop	globalPartnerships		add	0	0	0	
prop	econExpansion		add	0	0	0	
prop	publicEng		add	0	0	0	
TA	anytimeOutpost	anytime return for outpost?	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	0		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		
TA	anytimeIntO	anytime return for int outpost?	none	no	yes		
prop	devRisk		mult	1	0.95		
prop	humanCiv		add	0	0		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		
OP	outpostEnergyStorage	outpost energy storage	none	none	batteries	fuelcell	
prop	devRisk		mult	1	0.99	0.92	
prop	humanCiv		add	0	0	0	
prop	scienceKnowledge		add	0	0	0	
prop	marsPrep		add	0	0	1	
prop	globalPartnerships		add	0	0	0	
prop	econExpansion		add	0	0	0	
prop	publicEng		add	0	0	0	
OP	outpostPowerGen	outpost power generation	none	solar	DIPS	nucReactor	
prop	devRisk		mult	1	0.95	0.92	
prop	humanCiv		add	0	1	2	
prop	scienceKnowledge		add	0	1	1	
prop	marsPrep		add	0	1	1	
prop	globalPartnerships		add	0	0	0	
prop	econExpansion		add	0	1	1	
prop	publicEng		add	0	0	0	

Table 5-7: Lunar Outpost Separable Property Table 3/3.

cat	shortID	Decision	units	alt A	alt B	alt C	alt D
OP	ioEnergyStorage	IntOenergy storage	none	NA	none	batteries	fuelcell
prop	devRisk		mult	1	1	0.95	0.92
prop	humanCiv		add	0	0	0	0
prop	scienceKnowledge		add	0	0	0	0
prop	marsPrep		add	0	0	0	1
prop	globalPartnerships		add	0	0	0	0
prop	econExpansion		add	0	0	0	0
prop	publicEng		add	0	0	0	0
OP	ioPowerGen	IntOpower generation	none	NA	solar	DIPS	
prop	devRisk		mult	1	1	0.95	
prop	humanCiv		add	0	0	1	
prop	scienceKnowledge		add	0	0	1	
prop	marsPrep		add	0	0	1	
prop	globalPartnerships		add	0	0	0	
prop	econExpansion		add	0	0	1	
prop	publicEng		add	0	0	0	
OP	nuclearPolicy	nuclear policy	none	none	rtg_like	rtg_like_or_fission	
prop	devRisk		mult	1	1	1	
prop	humanCiv		add	1	0	0	
prop	scienceKnowledge		add	0	0	0	
prop	marsPrep		add	0	0	0	
prop	globalPartnerships		add	0	0	0	
prop	econExpansion		add	0	0	0	
prop	publicEng		add	0	0	0	
CM	commRepeater	is there a repeater?	none	no	yes		
prop	devRisk		mult	0.99	1		
prop	humanCiv		add	0	0		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		
CM	dataRate	communications data rate	none	telemAndVideo	HDLive		
prop	devRisk		mult	1	1		
prop	humanCiv		add	0	0		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	1		
prop	publicEng		add	0	1		
CM	LOItelemetryReq	is LOI telemetry required?	none	no	yes		
prop	devRisk		mult	1	1		
prop	humanCiv		add	0	0		
prop	scienceKnowledge		add	0	0		
prop	marsPrep		add	0	0		
prop	globalPartnerships		add	0	0		
prop	econExpansion		add	0	0		
prop	publicEng		add	0	0		

5.3.8 Lunar Outpost Architecture Decision Graph

In previous chapters, we have shown the complete view, the logical view, and the properties view of the ADG. In this chapter, the logical view is especially of interest. The complete view and property view are shown in Appendix B.

The logical view of the architecture decision graph for lunar decision problem is shown in Figure 5-5. This particular logical view illustrates an interesting property of the decision problem formulation for the lunar outpost architecture problem. The visualization of the ADG clearly shows that it is composed of two logically independent subgraphs. They can be considered logically independent since the two subgraphs are not connected by any logical constraints.

The implication of the two logically independent subgraphs is that there are two subsets of decisions which are not logically connected to each other. Since all property functions in this study are additive or multiplicative separable functions (see 3.3.1), this decision problem can be divided into two separate subproblems.

Figure 5-5 identifies these two subproblems as Part A and Part B. Part A, is composed entirely of decisions from the categories of campaign strategy (CS) and human habitation (HH). Part B is composed of problems decisions from the categories of communications (CM), outpost power (OP), and surface mobility (SM).

By dividing up the problem into two separate ADG decision analysis problem, we can save on computational effort, since the computational effort scales exponentially with the number of decision variables (see Section 3.7.2). Since the decisions in Part A do not depend on the decisions in Part B, the logical constraints between these decisions can be evaluated separately. Furthermore, since all seven property functions used in this study are separable property functions, they can be evaluated separately for Part A and Part B.

The visual inspection of the structure of the ADG can be considered a form of structural reasoning. This a similar to the discussion in Sections 4.2.6 and 4.3 for the Apollo problem, where certain disconnected parts of the problem were identified. The Apollo problem could not be separated because its decision variables were connected by non-separable properties. But, in the case of the Lunar Outpost problem, we can take this structural reasoning information a step further and use it divide the problem into two computational problems, since the property functions for the lunar outpost ADG are all separable functions. Each of the two subproblem requires fewer computational resources, since the computational resource requirements grow exponentially with the number of decision variables in the ADG (see Section 3.7.2).

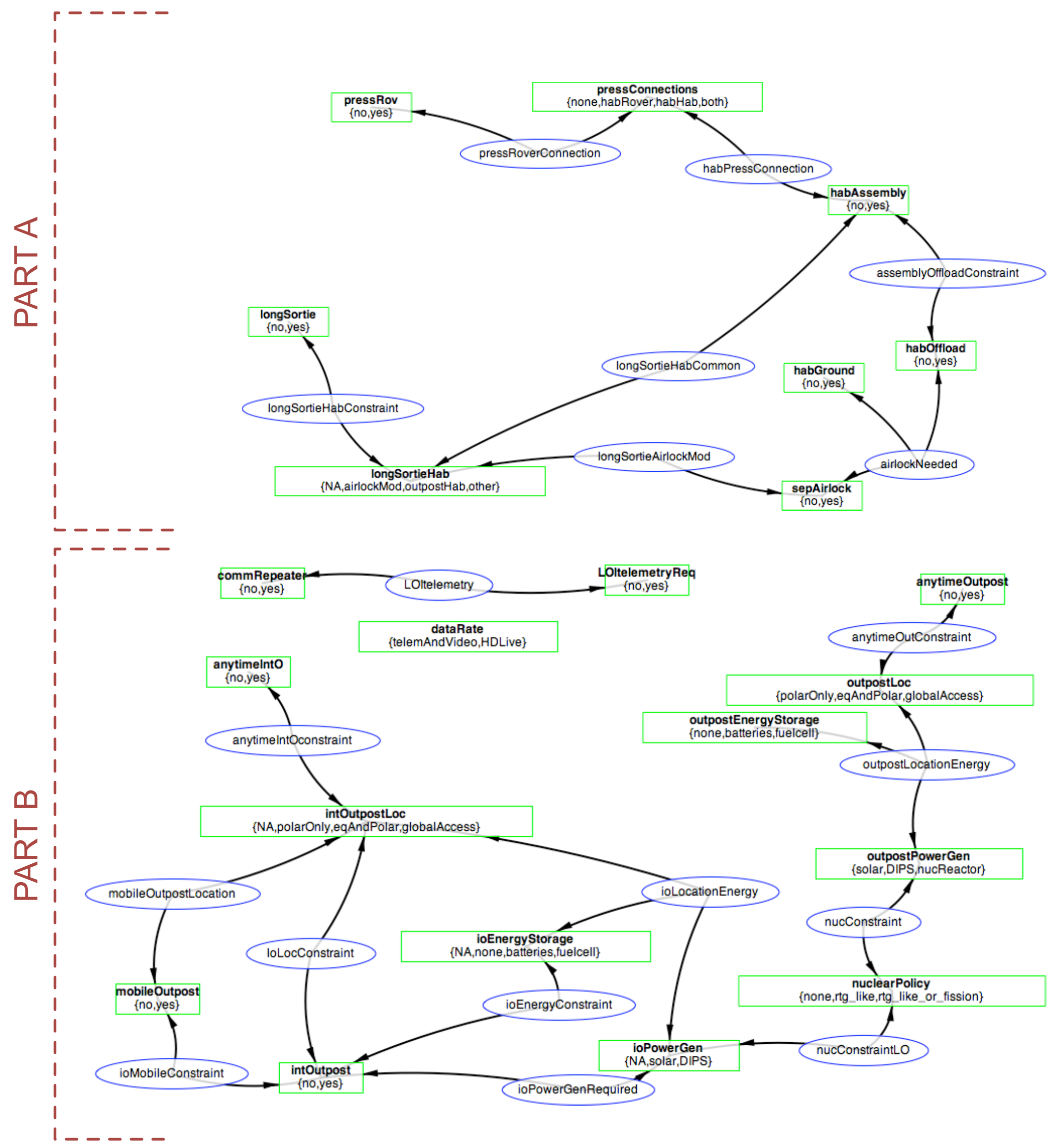


Figure 5-5: Lunar outpost ADG – logical view. Decision variables are green rectangles and logical constraints are blue ellipses. Note that property variables and property functions are not shown in the logical view of an ADG (see Section 3.3.2. Visual Inspection of the logical view immediately indicates that the logical constraints for the decision problem can be divided into two categories which are not connected though logical constraints: Part A (Human Habitation and Surface Mobility) and Part B (Transportation, Communications, Power, and Surface Mobility).

5.4 Structural Reasoning

The structural reasoning process is the second process in the ADG cycle (Figure 3-1). It involves analyzing the structure of the ADG and producing structural information about the decision problem. It was pointed out that visual inspection of the ADG is a form of structural reasoning. This section calculates some formal structural properties of the decision variables by applying the two structural reasoning algorithms, ADGsort1 and ADGsort2, introduced in Section 3.4 to each of the two parts of the ADG.

The results of ADGsort1 and ADGsort2 are presented in Tables 5-8 and 5-9. ADGsort1 determines the degree of connectivity between decisons. This is a first order measure of how many decisions are directly connected through adjacent logical constraints. ADGsort2 is an alternative measure of the structural properties of the ADG which adjusts the degree of connectivity by subtracting the size of each decision variable's set of alternatives. Specific descriptions of these algorithms are contained in Section 3.4.

Table 5-8: Results of ADGsort1 for the outpost decision problem.

decision	deg. of conn.	decision	deg. of conn.
pressConnections	2	mobileOutpost	2
sepAirlock	3	LOItelemetryReq	1
longSortie	1	intOutpost	4
longSortieHab	3	dataRate	0
pressRov	1	intOutpostLoc	5
habOffload	3	anytimeOutpost	1
habAssembly	3	outpostLoc	3
habGround	2	ioEnergyStorage	3
		anytimeIntO	1
		commRepeater	1
		ioPowerGen	4
		outpostEnergyStorage	2
		nuclearPolicy	2
		outpostPowerGen	3

The primary purpose of applying the structural reasoning algorithms to the ADG is to provide a heuristic to enhance simulation performance. The values for either degree of connectivity or sort2rank are used to guide the construction of an executable OPN model. The impact of structural reasoning results on the efficiency of decision problem simulation is discussed in Section 5.7.1.

Table 5-9: Results of ADGsort2 for the outpost decision problem.

decision	**sort2rank**	**decision**	**sort2rank**
pressConnections	-2	mobileOutpost	0
sepAirlock	1	LOItelemetryReq	-1
longSortie	-1	intOutpost	2
longSortieHab	-1	dataRate	-2
pressRov	-1	intOutpostLoc	1
habOffload	1	anytimeOutpost	-1
habAssembly	1	outpostLoc	0
habGround	0	ioEnergyStorage	-1
		anytimeIntO	-1
		commRepeater	-1
		ioPowerGen	1
		outpostEnergyStorage	-1
		nuclearPolicy	-1
		outpostPowerGen	0

From a systems engineering management perspective, the results can also be used for task prioritization and scheduling. A high ranking in terms of degree of connectivity indicates that a decision is highly connected. In this example, the intermediate outpost location (intOutpostLoc) decision variable is connected to five other decisions through logical constraints (degree of connectivity=5). This indicates that a change in the intOutpostLoc decision variable may impact the set of available alternatives for at least five other decision variables. From the ADG in Figure 5-5, we can see that these five decision variables are: ioEnergyStorage, ioPowerGen, anytimeIntO, mobileOutpost, and intOutpost. On the other extreme, the communications data rate decision variable (dataRate) is not logically connected to other decision variables (degree of connectivity=0). A change in communications data rate does not change the set of available alternatives for any other decisions.

This information derived from the degree of connectivity is useful to an engineering manager. It implies that a team considering the intermediate outpost location decision variable must collaborate with any other teams considering the five other connected decision variables. On the other extreme, a team considering the dataRate decision variable can choose one of its alternatives without considering how it would impact the available alternatives for the other architectural decision variables.

From a higher-level point of view, the structural reasoning results indicate that two separate engineering groups could consider the decisions in Part A separately from the decisions in Part B. The two groups could study the feasibility of different combinations of alternatives without the need to coordinate their sets of decision variable assignments with the other team.

5.5 Simulation

The simulating process is the third process in the ADG cycle (Figure 3-1). It involves transforming the structural information about the ADG into an executable OPN model and then using the OPN model to produce a set of feasible combinations of decision variable assignments and their properties, $\mathcal{C}$. The algorithm for completing this task, OPNbuild1, is described in detail in Section 3.5. Since this ADG was separated into two logically disconnected parts, simulation of the ADG was completed by building two separate OPNs using the OPNbuild1 algorithm (Algorithm 3.3). The two OPN models corresponded to Part A and Part B of the ADG[4]. OPNbuild1 requires a decision variable ordering as one of its inputs. For the lunar outpost decision problem, the results of ADGsort2 were used (Table 5-9).

The result of the simulation is an data set consisting of the feasible combinations of assignments to the decision variables and the value of their properties, which satisfy all constraints. For part A, 186 feasible combinations of decisions were produced out of a total combinatorial space consisting of 1024 combinations of decision variable assignments. For part B, 47659 feasible combinations of decisions were produced out of a total combinatorial space consisting of 497664 combinations of decision variable assignments[5]. The results of the simulation are presented in the next section. The impact of the two different variable ordering algorithms on the efficiency of the simulation performance is discussed in Section 5.7.1.

5.6 Viewing

The fourth process of the ADG cycle (Figure 3-1) is decision data viewing. The viewing process transforms the feasible combinations of decision variable assignments and their properties, $\mathcal{C}$, into new available knowledge for the decision-maker. In this section, we use the decision space view plots to analyze the data.

[4]Visualizations of the two OPN executable models are omitted from this chapter for brevity.

[5]The total size of the combinatorial space is calculated by suspending the logical constraints.

5.6.1 Decision Space Views

Since the problem was divided into two parts, different decision space views were created for part A and part B for each metrics. For each figure in this section, the decisions in Part A are shown in the left plot and the decisions for Part B are shown in the right plot. Definitions for each property variable and function were presented in Section 5.3.7. Property Variable Sensitivity (PVS) is defined in Section 3.6.1. PVS is a measure of the potential impact of a change in a decision variable on a system property. It is a measure of the magnitude of the shift in the property function's mean value over the entire set of feasible combinations of decisions due to a change in a decision variable assignment.

Development Risk

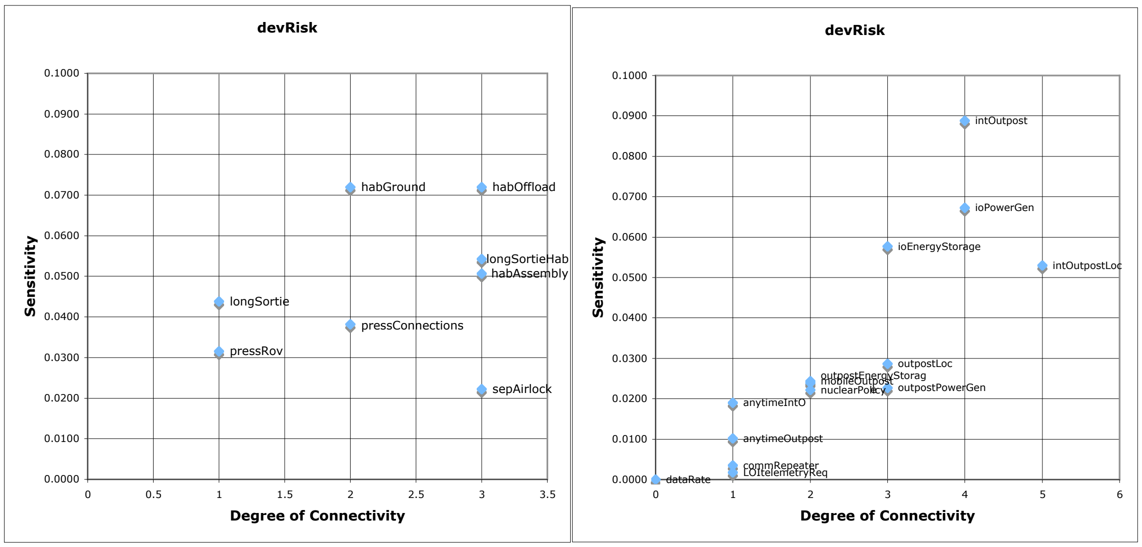


Figure 5-6: Decision Space View for the development risk property. Decisions in Part A are shown in the left plot,and decisions in Part B are shown in the right plot.

Figure 5-6 is the decision space view for the development risk property. The development risk property is a proximate measure for both the cost and difficulty of a campaign.

The plot on the left of Figure 5-6 refers part A. The decisions that are both strongly connected and sensitive are decisions about habitat ground access, habitat offloading, and habitat assembly. Intuitively, this makes sense since these decisions will define the interfaces between surface elements and, to some degree, the long-term capabilities of the outpost.

Note that the decisions about whether or not there will be pressurized rovers and long sorties (campaign element B) do not have a high degree of potential impact on development risk. This also makes intuitive sense, since these two decisions are related to enhancements to the architecture which could be added to the architecture at any time during the outpost's life cycle. This result agrees with the findings in the Lunar Architecture Update [NAS07], which concluded that a pressurized lunar rover could be added to any campaign architecture.

The plot on the right of Figure 5-6 refers to part B. In this group of decisions, the ones that are the most strongly connected and potentially impactful on development risk are the ones related to the intermediate outpost. Specifically, the intermediate outpost's existence, location, power generation method, and energy storage technology. Note that PVS measure does not indicate that including an intermediate outpost will increase development risk, it indicates that changing the decisions about the intermediate outpost can effect development risk. This change can be either up or down.

Mars Preparation

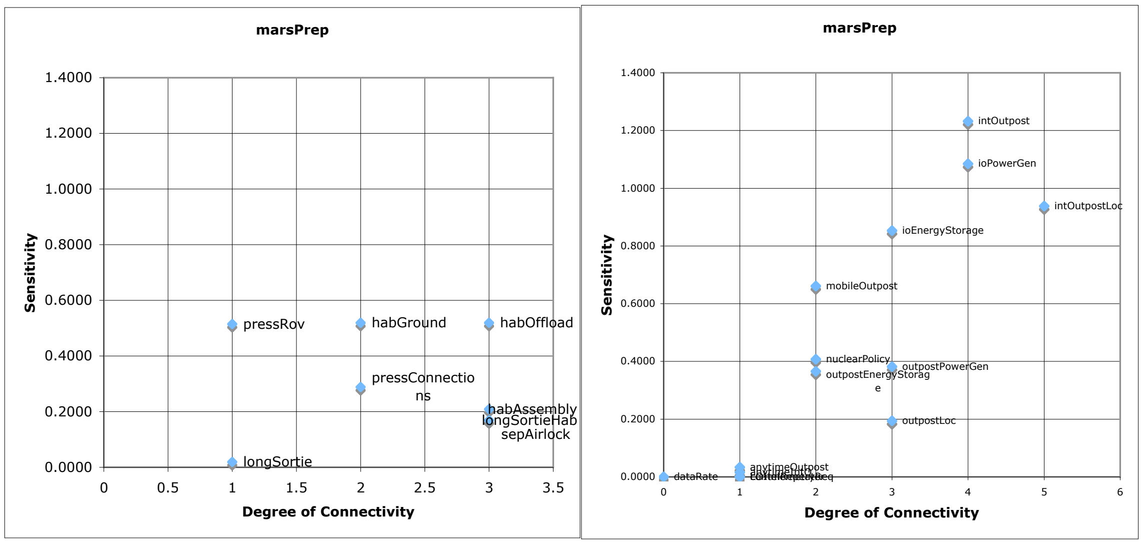


Figure 5-7: Decision Space View for the Mars preparation property. Decisions in Part A are shown in the left plot,and decisions in Part B are shown in the right plot.

Figure 5-7 is the decision space view for the Mars exploration preparation property. This property measures how a particular set of decisions may improve NASA's preparation for human exploration of Mars. It is an estimated version of the Mars Exploration Readiness Level metric presented in Reference [Kle05].

For Part A, decisions about habitat ground access requirements and habitat offloading are both highly impactful in terms of connectivity to other decisions and impact on the Mars preparation property. This is because in the case of Mars exploration, it is more likely that the Mars surface habitat will either be a pre-deployed, integrated habitat or double as the Earth to Mars in-space habitat. Habitat offloading, assembly, and direct ground access restrictions introduce operational complications that would be undesirable on a Mars exploration mission due to difficulties in aborting the mission.[WDR93, Hof04, ZBG91, Dra05].

For Part B, the most sensitive and strongly connected decisions are related to the lunar intermediate outpost: whether or not it exists, the power generation method, the energy storage technology, and the intermediate outpost location. Following the argument above, this makes intuitive sense, since the intermediate outpost is an integrated outpost that is delivered to the lunar surface on one uncrewed lander. This type of habitat design is more similar to the one that could be used for Mars exploration[Dra05, Zub97, WDR93].

Additionally, the intermediate outpost decision impacts Mars preparation because the study used the modeling assumption that an intermediate outpost could be deployed earlier in the program at lower cost. This would provide earlier surface EVA experience, resulting in useful data during the planning stages of Mars exploration effort.

Of note is the position of the nuclear policy decision in the decision space view for Part B. It is in the category of weakly connected and insensitive decision variables (lower left quadrant). This results is aligned with the assumptions of previous studies, which have shown that nuclear power generation is helpful, but not required for Mars exploration[Dra05, Woo07].

Public Engagement

Figure 5-8 is the decision space view for the public engagement property. This property measures how a particular set of decisions may help "engage the public, encourage students and develop a high-tech workforce".

For Part A, the strongly connected and sensitive decisions include decisions about the type of long sortie habitat and pressurized connections. Decisions about pressur-

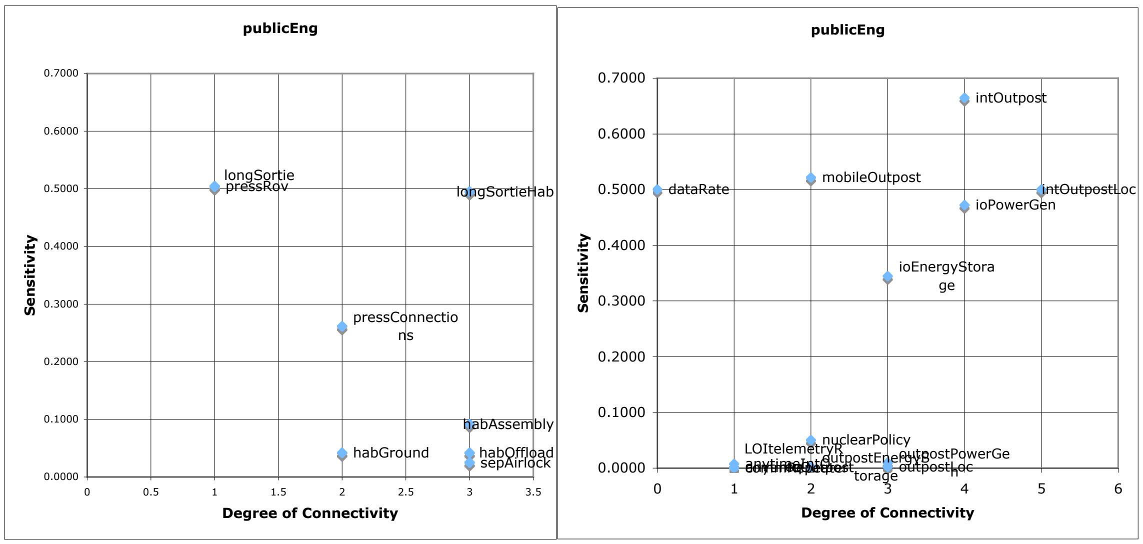


Figure 5-8: Decision Space View for the public engagement property. Decisions in Part A are shown in the left plot,and decisions in Part B are shown in the right plot.

ized rovers and long sorties are also relatively high on the property variable sensitivity axis, but not on the degree of connectivity axis. These decision variables are likely to affect increased public engagement in the lunar exploration program because they affect the number of interesting events in the exploration program. Long sortie missions and pressurized rovers will allow astronauts to visit interesting lunar sites more often. If the architecture includes pressurized connections, it may imply that there are more habitat assembly events, which may catch the public's interest.

For Part B, the decision variables which are the most highly connected and have the highest degree of property variables sensitivity are the intermediate outpost decisions. An architecture that includes an intermediate outpost allows for earlier exploration in comparison to one that relies on one or a few long-term outposts. Earlier exploration return will give NASA earlier opportunities to engage the public in the lunar exploration program. Additionally, the idea of a mobile intermediate outpost has the potential to allow more events of public interest because the outpost could move around the surface of the moon between crew visits.

Also in Part B, the data rate decision has the potential to affect public engagement. By providing live high definition television signals from the surface of the moon, members of the public could be actively engaged in the activities on the moon. In particular, high-speed bi-directional communication could create opportunities for direct interaction between educational groups and the astronauts.

Global Partnerships

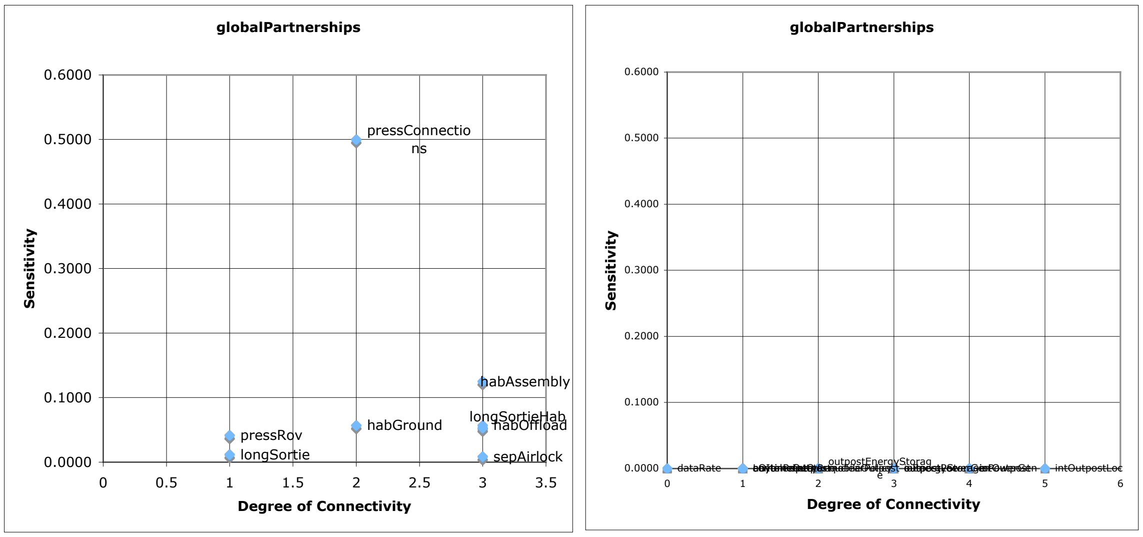


Figure 5-9: Decision Space View for the global partnerships property. Decisions in Part A are shown in the left plot,and decisions in Part B are shown in the right plot.

Figure 5-9 is the decision space view for the global partnerships property. This property measures how a particular set of decisions may help "provide a challenging, shared and peaceful activity that unites nations in pursuit of common goals".

There is only one decision that stands out in the decision space view for this metric. It is the decision about what types of pressurized connections will be used. In the separable property Table (see Table 5-5 on page 144), choosing alternatives C or D allows for pressurized connections between habitats. By choosing this configuration, global participation may increase since foreign space agencies could build separate habitat modules and connect them to NASA's outpost modules.

Other decisions do not rank highly in this metric since none of them have a direct impact on the ability to provide opportunities for global participation. Global participation could take place at the module level, like it is done for the International Space Station, or at the technology development, equipment supply, or scientific analysis level. The decision variables in this study, besides the decision variable for pressurized connections, do not enhance or limit the opportunities for global participation in any particular way.

Economic Expansion

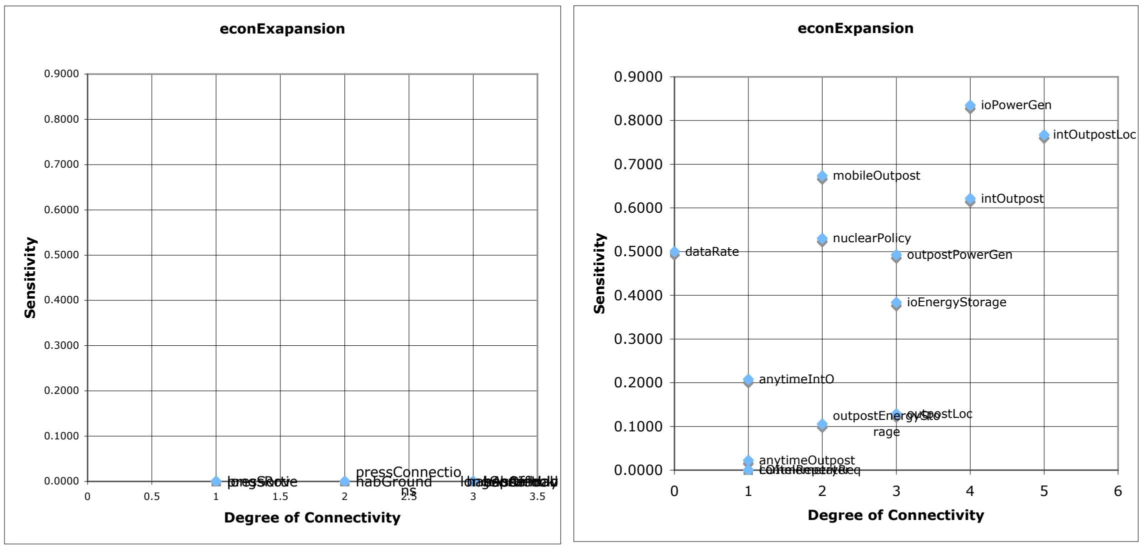


Figure 5-10: Decision Space View for the economic expansion property. Decisions in Part A are shown in the left plot, and decisions in Part B are shown in the right plot.

Figure 5-10 is the decision space view for the economic expansion property. This property measures how a particular set of decisions may help "expand earth's economic sphere, and conduct lunar activities with benefits to life on the home planet."

The decisions in Part A, which are in the decision categories of campaign strategy and human habitation do not have a strong impact on economic expansion. None of these decision variables, which are in the categories of human habitation and surface mobility have a direct connection to economic expansion. However, the decisions in Part B, which are in the categories of transportation, communications, power, and surface mobility, have some impact. Specifically, the decision variables that are related to long-term and intermediate outpost locations on the lunar surface and power generation technologies are sensitive and strongly connected. This is due to an assumption in the economic expansion property function that greater coverage of the lunar surface and readily available power would provide benefits to economic activity on the moon, such as mining or space tourism.

Science Knowledge

Figure 5-11 is the decision space view for the science knowledge property. This property measures how a particular set of decisions may help "pursue scientific activities that address fundamental questions about the history of the Earth, the solar system, and the universe."

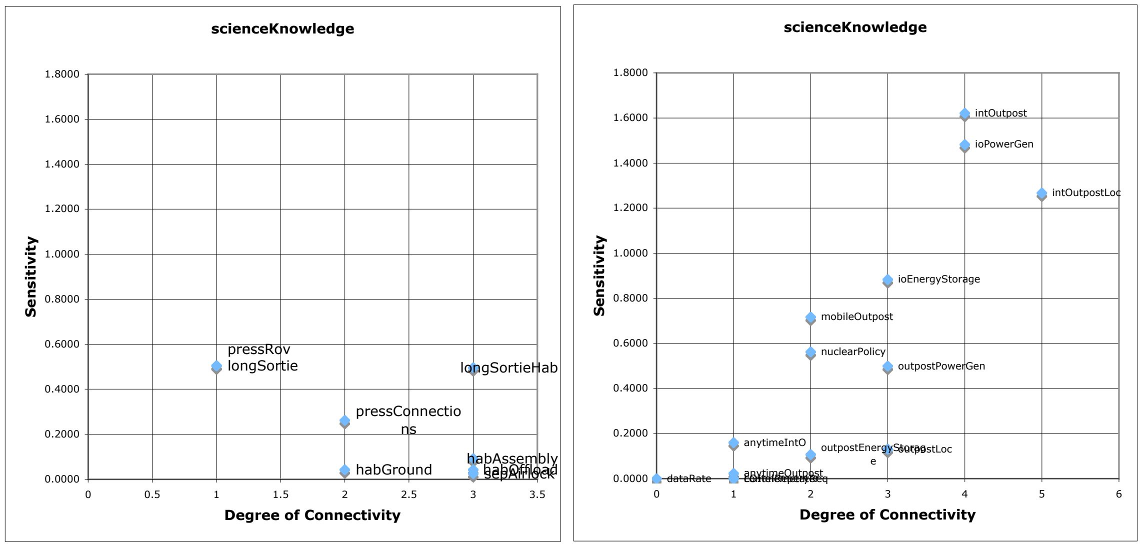


Figure 5-11: Decision Space View for the science knowledge property. Decisions in Part A are shown in the left plot, and decisions in Part B are shown in the right plot.

For the science knowledge property the decisions which have the greatest potential impact are the decisions about the pressurized rover, the long sortie, the long sortie habitat, the existence of intermediate outpost, the intermediate outpost's power generation and the intermediate outpost's location. The decisions about the intermediate outpost energy storage technology, pressurized connections and the existence of a mobile outpost are also impactful, in terms of property variable sensitivity, but not as strongly connected.

In general, the decision variables that have the biggest affect on science knowledge are generally the ones which affect the number of sites and rate of scientific data gathering. In particular the existence of long sorties, pressurized rovers, and intermediate outposts as well as global access to the lunar surface increase the scientific knowledge property.

Human Civilization

Figure 5-12 is the decision space view for the human civilization property. This property measures how a particular set of decisions may help "extend human presence to the Moon to enable eventual settlement."

For Part A, the most sensitive and strongly connected decisions are in regards to the habitat ground access requirement, habitat offloading, habitat assembly, and pressurized connections. These decisions are sensitive since the technology development

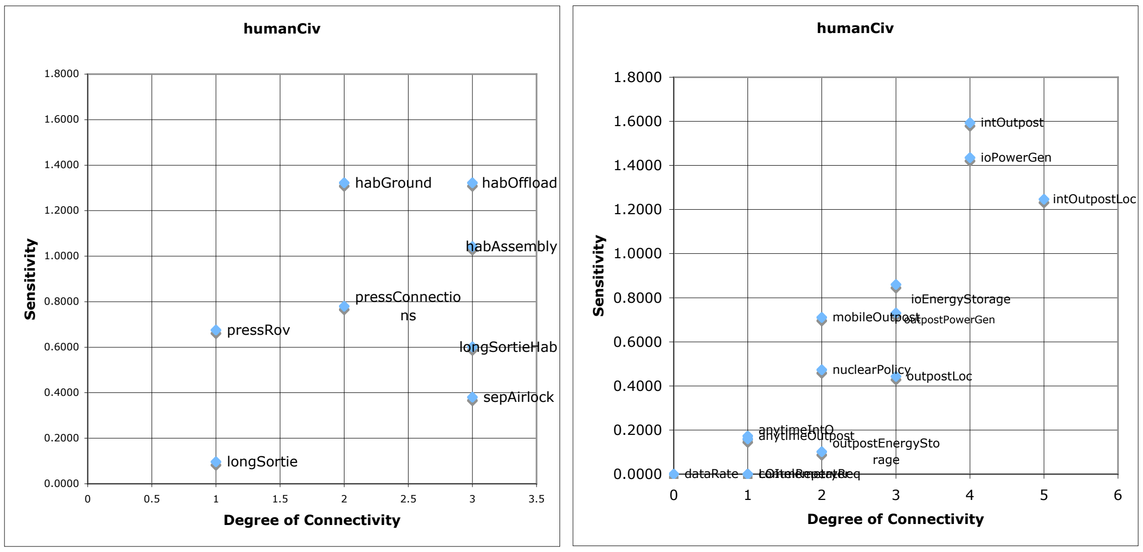


Figure 5-12: Decision Space View for the civilization property. Decisions in Part A are shown in the left plot, and decisions in Part B are shown in the right plot.

and experience gained through offloading and assembling habitats with direct surface access is applicable to long-term lunar settlement goals. A long term settlement is more likely to be constructed of many habitat units, rather than single monolithic outpost habitats.

For Part B, the more sensitive and strongly connected decisions are the decisions regarding the intermediate and long-term outpost locations, power generation, and energy storage technology. Certain alternatives for these decision variables, such as nuclear power generation or global surface access, could help develop the technology and experience necessary for eventual human settlement. Deploying intermediate outposts could provide earlier experience living on an extra-terrestrial planetary body, which could then benefit the long-term technology development goals for a permanent settlement.

5.7 Analysis of the Methodology

5.7.1 Impact of the Structural Reasoning Algorithms on Simulation Performance

The performance of the sorting algorithms is measured by their impact on the efficiency of the simulation algorithm. The lunar outpost study provides the opportunity to measure the efficiency impact on ADGsort1 and ADGsort2 on two more ADG's (one for part A and one for Part B). We follow the same methodology to test structural reasoning impact as we did in Section 4.6.1 of the Apollo study. Since the number of possible orderings was very large, we use a random sample of variable orderings rather than a complete enumeration of all possible orderings.

The performance test for Part A is shown in Figure 5-13 and the performance test for Part B is shown in Figure 5-14. For part A, which has a smaller number of decision variables, a larger sample set of 298 random variable orderings was used. For Part B, which has a large number of decision variables, a smaller sample set of 30 random orderings was used. This is due to the exponential increase in computation time required to complete the analysis of Part B's ADG. This test was run on Windows 2003 sever using an Intel Core 2 Duo 6600 Processor (2.40GHz) with 4 GB of ram. The Java JVM was version 1.6.

The data in Figures 5-13 and 5-14 shows some similar features to the data from the previous performance tests in Figure 3-14 (the example ADG problem) and 4-10 (the Apollo Study). Both figures show a correlation between the number of tokens generated and the total computational time. For Part B, which has 14 decision variables, the ordering produced by ADGsort1 and ADGsort2 perform within the top 10% of the best ordering in the sample set. For Part A, the ADG has only 8 decision variables. The performance of ADGsort2 is among the fastest variable orders in the sample set, however the variable order due to ADGsort1 is only in the top 50% of variable orderings.

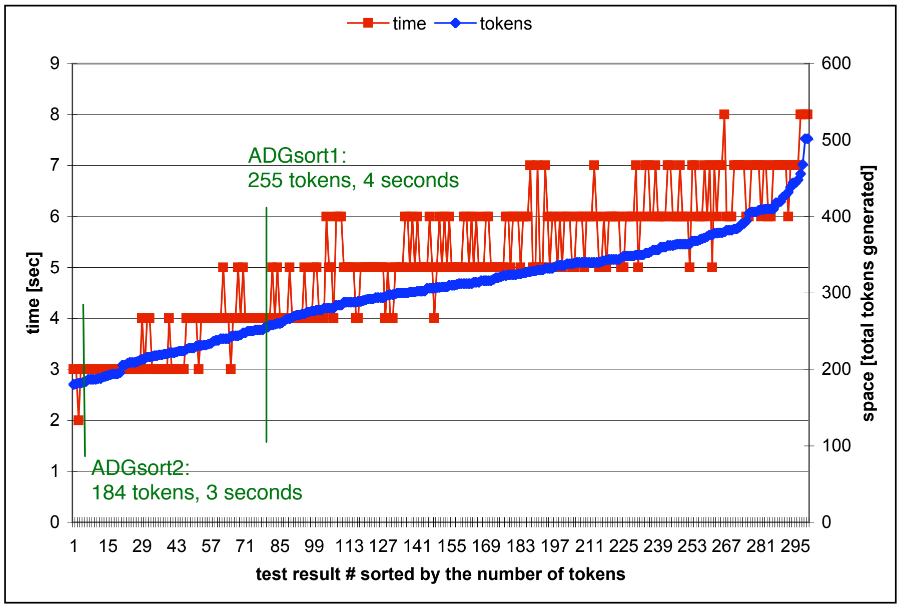


Figure 5-13: OPNBuild1 Performance for the Outpost Problem, Part A. This test of simulation performance includes the orderings produced by ADGsort1 and ADGsort2 as well as 298 other random decision variable orderings. This is a 2-axis vertical plot, time [sec] and space [total tokens generated]. The performance of the orderings produced by ADGsort1 and ADGsort2 are indicated with the green vertical lines.

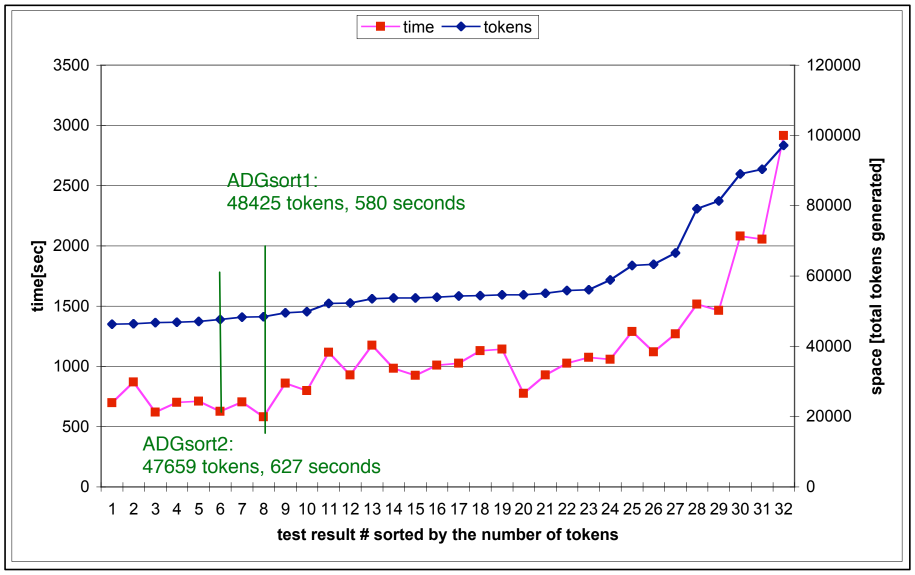


Figure 5-14: OPNBuild1 Performance for the Outpost Problem, Part B. This test of simulation performance includes the orderings produced by ADGsort1 and ADGsort2 as well as 30 other random decision variable orderings. This is a 2-axis vertical plot, time [sec] and space [total tokens generated]. The performance of the orderings produced by ADGsort1 and ADGsort2 are indicated with the green vertical lines.

5.8 Summary

This chapter studied the architecture for NASA's lunar exploration program using the ADG methodology. The four guiding principles, first presented in Section 4.2.1 were used to transform the architecture problem into a decision problem.

The architecture space was parameterized using a set of decisions related to the existence of certain campaign elements [HWC07], their locations, and their technology configurations. Two policy decisions were included in the decision model: nuclear policy and anytime return requirements. High-level decisions about the communications system and surface mobility were also included in the decision model. Logical Constraints encoding the feasibility of combinations of alternatives for each decision variables were developed.

Seven property functions were developed. Six of them were aligned with NASA's six "guiding themes" for the lunar exploration strategy[NAS06b, Tea07, CN06]: Human Civilization, Scientific Knowledge, Exploration Preparation, Global Partnerships, Economic Expansion, and Public Engagement. The seventh property function was development risk, a proxy metric for cost and risk.

The decisions, logical constraints, property variables, and property functions were encoded in an ADG. Once the ADG was created the logical view made it evident that the decision problem could be divided into two separate sub-problems. Since all of the property functions used in this study were either multiplicatively or additively separable functions, it was possible to split the set of decisions into two sub-sets. The two sets of decisions were called Part A and Part B. Part A included decisions from the categories of campaign strategy (CS) and human habitation (HH). Part B included decisions from the categories of communications (CM), outpost power (OP), and surface mobility (SM).

The simulation of the two ADGs was used to create the set of feasible combinations of decisions and calculate the property value sensitivities for each one. These results were plotted in decision space views. The performance of the simulation algorithm was tested using the two decision variable sorting algorithms, as well as a sample of random variable orderings.

From an engineering perspective, the ADG analysis of the lunar exploration program contributes new insight for the systems architecting effort. From structural reasoning of the logical view of the ADG alone, we can conclude that the set of decisions in the lunar outpost problem are logically decoupled (Section 5.4). Decoupling these two problems allows engineers to work on these decisions separately. The two groups of decisions split up the decision variables in to two groups:

- Group A: Decisions related to habitat configurations and types, pressurized connections, ground access, assembly, and offloading, and pressurized rovers.
- Group B: Decisions related to nuclear policy, power generation, energy storage technologies, intermediate outposts, intermediate and long-term outpost locations, communications, and anytime return requirements.

The decisions about the intermediate outpost's existence and related decisions are impactful in terms of nearly all property functions used in this study. They are impactful in the sense that they are the most logically connected to other decisions and they have a strong potential impact on system properties, as measured by the property variable sensitivity for development risk, Mars preparation benefit, public engagement, economic expansion benefit, science knowledge benefit, and human civilization benefit.

Decision variables regarding habitat offloading, assembly, and ground access are moderately coupled compared to all other decision variables in the ADG as measured by the degree of connectivity. In terms of some of the metrics they have relatively high property value sensitivities. Specifically, these decisions may be impactful on development risk, Mars preparation benefit, and human civilization benefit.

Decision variables about the communications configuration are largely decoupled from the other decisions. They are only logically connected within the subset of other communications decisions. The selection of the campaign elements does not depend on the selection of communications technologies and data rates. However, the decision variable about the data rate, in particular, has a strong potential effect on public engagement because it may provide a bi-directional gateway for the public to be connected to the astronauts on the lunar surface.

Two decisions that are not strongly connected to other decisions, but are impactful on the system properties are the decisions regarding the long sortie and pressurized rovers. These two decisions are impactful in terms of property variable sensitivity for Mars exploration preparation, public engagement, scientific knowledge, and human civilization. Including or excluding long sorties and pressurized rovers does not affect the feasibility of other architecture options. However, it may affect the number of sites that can be visited, the frequency of lunar activities of public interest, and the variety and scope of surface exploration experience from the lunar exploration program.

The aerospace engineering objective of this study was to identify the most potentially impactful architectural decisions for the lunar exploration program. The following conclusions can be drawn from this study:

- NASA should carefully analyze intermediate outpost decisions. The intermediate outpost is highly connected and impactful on property metrics because it is a potential low-initial cost option to provide early exploration returns.

- NASA should consider dividing engineering analysis teams into groups aligned with the decision variables from Part A and Part B.

- NASA should carefully consider the tradeoffs between the long-term human civilization benefits, development risks, and other system properties regarding the decisions about habitat assembly and ground access.

- NASA should strongly consider the decision variables for including or excluding long sorties and pressurized rovers, since they both provide benefit in terms of public engagement and science knowledge. However, since these decision variables are not strongly connected to other decision variables, these decisions could be delayed until after other decisions, like the intermediate outpost, power generation, and habitat configuration decisions are made.

- NASA should consider decisions about the communications systems independently of other decision variables. Although a communications systems will require a power source, there is connectivity, in terms of logical constraints between the communications system and other parts of the architecture. A change in the configuration of the communications system will not impact the feasibility of other architectural decisions.

6 Conclusions

6.1 Discussion

Decisions in systems architecture are actions that lock down the properties, limit the functions, limit the forms, or change the concept of a system in order to better meet the needs and goals of a system's stakeholders. Decision-making in system architecture is considered challenging since the decisions are generally non-routine, significantly consequential, interconnected, and often based on weakly-defined models of the system. Chapter 1 established that in practice, systems architects refine an architecture by making decisions. The objective of this thesis was to show that by creating an explicit representation of architecture as the result of a set of decisions, a systems architect can effectively reason about an architectural candidate space and gain useful guidance for the decision-making process.

Chapter 2 surveyed the decision support literature to determine which aspects of existing methods and tools could be leveraged to meet these objectives. Three of these methods and tools, the Morphological Method (Section 2.3.1), Constraint Satisfaction Problems (Section 2.3.3), and Object Process Network (Section 2.3.4) significantly influenced the research.

Chapter 3 introduced the Architecture Decision Graph (ADG) framework. The ADG framework represents an architectural candidate space as a interconnected set of decision variables, logical constraints, property variables, and property functions. The framework leverages this representation to transform the architecture problem into a computation problem, generate feasible combinations of decisions, determine their properties, and present decision support information to the architect. The decision support information presented to the architect includes Pareto front views, which identify the properties of the set of non-dominated solutions, and decision space views, which identify the degree of impact for the architecture's decision variables. The usefulness of the ADG framework was demonstrated through two case studies: a retrospective study of architectural decisions for the Apollo project of the 1960's (Chapter 4) and a study of architectural decisions for NASA lunar outpost architecture (Chapter 5).

In the retrospective study of Apollo, it was demonstrated that the most important decisions were recovered through ADG's methodology. Specifically, the decision about Lunar Orbit Rendezvous (yes or no) is clearly identified as a decision that stands out as highly impactful in the decision space views for mass and risk presented in Figure 4-9. Historical records [Sea96, BGS79, Han95, MC04, EMB$^+$78, Han99] and an interview with the chief architect of the Apollo program, Dr. Robert Seamans [SKK05] concur that the Lunar Orbit Rendezvous mission mode decision was the most critical and consequential decision for the entire Apollo project. Furthermore, ADG analysis identified lander crew size and lander fuel type as the next most impactful decisions. This concurs with the 1961 Houbolt report[Hou61a], which asserts that the three decisions LOR (yes or no), lander fuel type, and lander crew size were the most impactful decisions for the 1960's lunar exploration architecture.

The lunar outpost study in Chapter 5 demonstrated that ADG can be used to analyze a novel architecting problem. The lunar outpost project is an ongoing effort by NASA to return astronauts to the moon and establish a long-term lunar outpost [Ald04]. The application of ADG to this architecting problem was able to identify important features of this decision problem and identify the decision variables which have the highest potential impact on the architecture.

Through the application of structural reasoning to the outpost problem, the ADG framework was able to identify two decoupled groups of decision variables:

1. Decisions involving habitats, pressurized connections, ground access, assembly & offloading, and pressurized rovers.

2. Decisions involving power generation, energy storage, locations, communications, and anytime return requirements.

The identification of these two logically decoupled groups is significant from an engineering task management perspective because it suggests that these two groups of decision variables could be studied by two different engineering teams. Since changes in one of these two groups of decision variables do not affect the feasibility of alternatives for the other group, the engineering teams could study the two parts of the problem independently.

The lunar outpost study also included decision space views, which identified the relative degree of impact for each decision variable in terms of its degree of connectivity and property variable sensitivity. A detailed discussion of the insight gained in the lunar outpost study through the use of the ADG framework is contained in Section 5.8. The following list is an example of the types of specific conclusions that were made:

- Decisions about the intermediate outpost's existence and other intermediate outpost related decisions are highly coupled with other decisions in the model and have a potentially strong impact on system properties relative to other decisions in the model.

- Decisions about communications are largely decoupled from the other decisions.

- Decisions about habitat offloading, assembly, and ground access are also moderately coupled, and in some cases, have strong effect on system properties.

- Including of a pressurized rover in the lunar exploration program is helpful for Mars preparation, public engagement, scientific knowledge, and human civilization. The decision variable about the pressurized rover is largely decoupled from the rest of the architecture.

- Including a long sortie mission potentially impacts public engagement, science knowledge, but could be detrimental for development risk. The decision variable about the long sortie mission is also largely decoupled from the rest of the architecture.

These conclusions supplement the other on-going analyses of the lunar outpost program [HWC07, HSWC07, NAS06d, NAS07]. Future developments in the lunar exploration program will determine whether or not these decision variables identified by the ADG framework were truly the driving decision variables in practice.

6.2 Contributions

The goals of this research were: Develop a way to explicitly represent systems architectures as the result of a set of interconnected decisions; Develop a tool to leverage this representation in order to transform an architectural problem in a computational problem; and Develop views of the architectural decision space that provide guidance for architectural decision-making. The contributions of the ADG framework presented in this thesis are aligned with these three goals:

- **Contribution 1:** Architecture Decision Graph (ADG), the representational aspect of the ADG framework, is a representation of an architecture candidate space as a set of interconnected decision variables, logical constraints, property variables, and property functions.
- **Contribution 2**: The structural reasoning and simulation aspects of the ADG framework provide the tools to turn an architectural decision problem into a computational problem.
- **Contribution 3**: The viewing aspect of the ADG framework provides a visualization of the degree of impact of decision variables in an architecture candidate space that can be used to inform engineering task breakdown.

6.3 Features of the ADG Framework

Chapter 3 of this thesis describes the Architecture Decision Graph (ADG) framework for decision support in systems architecting. It supports human decision making by providing a methodology and tool for analyzing architectures as a set of interrelated decisions. The features of ADG can be broken down into four categories:

1) ADG provides a explicit representation of architecture as a set of interrelated decisions.

ADG represents system architecting problems as a graph of decision variables, property variables, logical constraints, and property functions. Experience using ADG has shown that it is adequately expressive for solving system architecting problems (see the case studies, Chapters 4 and 5. It is especially well-suited for non-sequence dependent problems. Since ADG's simulation engine is based on OPN, it allows multiple types of property functions to be specified using OPN's Jython scripting interface.

2) ADG provides structural reasoning tools for systems architecting:

Since ADG consists of a small set of primitives and a graph structure it provides a computationally simple and consistent way to extract the degree of connectivity of decision variables in an architecting problem. Using the structural reasoning tools provides the ability to identify logically disconnected sub-problems within the architecture space (see Section 5.3.8). From an engineering management perspective, identifying logically disconnected sub-problems provides guidance for engineering task breakdown.

3) ADG provides simulation tools for identifying and simulating feasible sets of decisions

ADG's simulation layer enumerates and calculates properties for all feasible combinations of decision variables. In contrast to valued-CSP models, decision trees, influence diagrams, and sequential decision diagrams, more than one property function can be calculated for each element in the feasible set. The only restriction on the type of property function that can be used to calculate a system property in ADG is that it must be implementable in the Jython scripting language [PR02] and terminate in finite time.

Simulating a decision problem using ADG is achieved through an automatically generated OPN model. The OPN model is automatically generated from the information about the decision problem encoded in ADG's graph representation. The generated model is likely to be more efficient than one generated by hand, since the construction is sequenced by the ADG's structural reasoning heuristics. Performance tests in Chapters 3, 4, and 5 show that the OPN models are built using ADG's ordering are among the best 10% in terms of computational resource consumption.

In addition to identifying feasible combinations of decisions, simulation of a decision problem using ADG also identifies all logically infeasible alternatives for decisions. This information is useful since infeasible alternatives can be removed from consideration in subsequent analysis of the decision problem.

4) ADG provides the decision space view

The decision space view is a visualization of the potential impact that a change in a decision variable's assignment could have on a system architecture space. The degree of impact is calculated in two different ways. The first measure is the decision variable's degree of connectivity. This measures the degree of impact in terms of how many other decisions are logically connected to a particular decision. A change in a highly connected decision will potentially change the feasible alternatives in a greater number of connected decisions.

The second measure is the decision variable's Property Variable Sensitivity (PVS). The property variable sensitivity (PVS) is a measure of the potential impact that a change in each decision variable assignment could have on a system property. PVS can be considered an enhancement of main effects analysis since it can calculate over more than two "levels" of decision alternatives; it does not require the selection of a fixed reference design, and is only measured over the feasible set of solutions.

The decision space view allows a decision maker to determine which decisions are high or low in each measure of sensitivity. By placing these two measures on one plot, it is possible to extract a suggested priority order for decisions.

6.4 Future Work

The representational, structural reasoning, simulating, and viewing aspects of ADG could be improved by further research. Below is a list of ideas for future research in the development of ADG as architecting tool.

6.4.1 Representing

Cognitive psychology research asserts that an effective representation of a problem can affect the performance of human problem solvers [Lev00]. By improving the user-interface of ADG, its effectiveness as a decision support system could be increased. By increasing the speed of decision problem encoding, a decision-maker could compete the ADG cycle in Figure 3-1 in less time. Improvements to the human interface should follow guidelines from the field of cognitive psychology[WH00].

The currently available problem entry methods for ADG are based on text files generated from spreadsheet software or by directly modifying the ADG graph in the OPN-Integrated Design Environment [Koo05]. An improved interface could be a web form which inputs data into a relational database. By using a relational database, variables could be checked for consistency using standard database tools. In addition, results from ADG simulation could be stored in a database format to enable flexible queries of subsets of data.

Chapters 4 and 5 provide some guidance as to how to choose the “right” decisions to model in ADG. However, the material provided is not a complete answer to the question. Since computational resources are finite, a methodology for effectively selecting which decision to model and how to partition alternatives is an open research area.

Currently, ADG does not explicitly model hierarchical decisions. As shown in Chapter 5, this issue can be worked around by adding a constraint which forces an irrelevant decision to take the value “NA”. However, a more explicit way to model this would involve allowing certain decisions to be turned on or off, depending on the outcome of another decision. One idea for representing decisions like this is the hierarchical morphological matrix presented in Reference [Cra07].

Section 4.6.2 pointed out that the ADG framework is well-suited for analyzing an architecture configuration space which involves decision variables that are not sequence or time dependent. Time-Expanded Decision Network (TDN), mentioned in Section 2.3.2 of the literature review, is a method for investigating time-dependant decisions involving switching between candidate architectures. However, TDN lacks an automated method for developing the initial candidate architectures. The ADG framework could be used as a front-end for TDN in order to identify a set of feasible architectures and possibly to calculate the properties of their switching costs.

6.4.2 Structural Reasoning

Currently, the choice of sorting algorithms is user-input driven. Ideally, ADG should be able to recognize which types of problems are more likely to benefit from ADGsort1 or ADGsort2.

Other structural reasoning algorithms are possible, such as using join-tree decomposition to provide a backbone tree for a parallelized OPN executable model. Dechter and Mateescu[DM07] demonstrated that using join-tree decomposition as a structural reasoning algorithm can dramatically reduce computational resource consumption for constraint satisfaction problems. Their work showed that a constraint graph's computational time could be reduced from scaling exponentially by the number of decision variables to exponentially by the product of the tree width[1] and the natural log of the number of decision variables. Progress towards implementing join-tree decomposition as a structural reasoning algorithm for ADG was reported in Reference [SKC06].

[1]see [BM96] and [DM07] for definition of tree width

6.4.3 Simulation

Simulation of an ADG currently does not allow a property variable to be connected to a logical constraint. However, allowing this type of connection would permit useful global decision pruning rules, such as "remove this set of decisions from the feasible set as soon as cost accumulates to greater than x".

Simulation time and space requirements might be further reduced by using a constraint propagation algorithm, similar to ones used in constraint graph problem solving [RN02]. Constraint propagation algorithms are used to detect infeasible decision alternatives by looking for inconsistencies with connect logical constraints.

6.4.4 Viewing

The simulation aspect of the ADG methodology produces a potentially large set of data. There are many potential ways this data could be plotted other than the two described in Section 3.6. One possibility is graphically representing a decision tree for a user-selected subset of decision variables. As mentioned in 2.3.2, decision trees are useful, clear representations of decision problems as long as they include a relatively small number of variables. A subset decision tree could show the branching relationship between two to five variables.

An interactive morphological matrix could be used to demonstrate the interconnectivity between decisions. After the feasible set of decisions is generated, a user could click on alternatives to enable or disable them. The interactive morphological matrix would respond by disabling alternatives for other decisions that become infeasible because of logical constraints. The interactive morphological matrix could be linked to decision space view plots that change as the set of alternatives is modified by the user.

6.5 Conclusions

The ADG framework is a generally applicable methodology for reasoning about a systems architecture. It is especially well-suited for architecting problems that involve determining the static configuration of the mapping of systems forms to system functions. Using the four principles for formulating an architecture problem as a decision problem (section 4.2.1), this thesis provides an architect with the guidance to parameterize an architectural candidate space into a set of interconnected decision variables. Through the application of the framework, systems architects have the ability to generate and evaluate instances of feasible architectures from a combinatorial space bounded by a set of decision variables.

This thesis is not the first to assert that the process of systems design is about making decisions (see [Cat06] and [Abr65], for example). There is general agreement that this assertion is true [DAE$^+$05]. However decision-based design methods are often criticized by design theorists [MR02, DAE$^+$05] since previous decision-based design methods have an assumption that the design process can only be enhanced by a decision-support framework after candidate designs are available. This means that these decision-based methods are only useful for additional refinement of designs that already exist. In contrast, the ADG framework does not assume that designs must exist before ADG is useful. Since ADG explicitly models architecture as the result of a set of decisions, candidate architectures can be generated by the ADG framework.

This thesis started with the goals to develop an explicit representation of architecture as a set of decisions, and to show that through using this representation, an architect can gain useful insight into the architectural candidate space. I conclude that by contributing the Architecture Decision Graph framework and developing views and methodologies for engineering task breakdown based on the potential impact of decisions, this thesis has achieved these goals.

Appendices

A
Supplemental Material for the Apollo Study

This appendix contains some additional information about the Apollo study presented in Chapter 4.

A.1 Apollo Property Functions

The Apollo architecture decision problem in Chapter 4 used two non-separable property functions to calculate the risk function and the IMLEO function. The Jython[PR02] source code for those two functions is listed below.

Listing A.1: ApolloPropertyFunctions.py

```
import math as m
#import synchronize as sync

dVeoa=9200*100/30
dVeod=11100
dVmoe=3840
dVdom=6798
dVmoa=7468
dvMod=3661
fs = 0.07
g=9.8*100/30;
Isp=311;
Isp1=315;
Isp2=425;

f=0.08;
frac_saturn = 0.12;

fi=0.1
eor_vehicle = 0;
lor_vehicle = 0;
lor_v_list = range(0)
eor_launch_count=0;
```

```
def cmMassCalc(cmCrew):
    if (cmCrew==2):
        return 8000;
    if (cmCrew==3):
        return 11000;
    else:
        return 0;

def lmMassCalc(lmCrew):
    if (lmCrew == 1):
        return 3000;
    if (lmCrew == 2):
        return 4000;
    if (lmCrew == 3):
        return 5000;
    else:
        return 0;

def sm4MassCalc(cmMass, Isp):
    #use Isp1 for now (storable)
    return StageMass(dvMod, Isp, cmMass, f);


def lm2MassCalc(LOR,lmMass, Isp):
    if (LOR == no):
        return 0;
    if (LOR == yes):
        return StageMass(dVmoa, Isp, lmMass, f);
    else:
        return 0;

def lm1MassCalc(LOR,lm2Mass,lmMass, Isp):
    if (LOR == no):
        return 0;
    if (LOR == yes):
        Mpay = lm2Mass + lmMass;
        return StageMass(dVdom, Isp, Mpay, f);
    else:
        return 0;

def lmTotalMassCalc(lm1Mass,lm2Mass,lmMass):
    return lm1Mass+ lm2Mass+ lmMass;

def sm3MassCalc(LOR, sm4Mass,cmMass, Isp):
    if (LOR == yes):
        return 0;
    if (LOR == no):
        Mpay = sm4Mass + cmMass;
        return StageMass(dVmoa, Isp, Mpay, f);

def sm2MassCalc(LOR, sm3Mass, sm4Mass, cmMass, Isp):
    if (LOR == yes):
        return 0;
    if (LOR == no):
        Mpay = sm3Mass + sm4Mass + cmMass;
        return StageMass(dVdom, Isp, Mpay, f);
    else:
        return 0;

def sm1MassCalc(lmTotalMass,sm2Mass,sm3Mass,sm4Mass,cmMass, Isp):
        Mpay = lmTotalMass+ sm2Mass +sm3Mass +sm4Mass +cmMass;
        return StageMass(dVmoe, Isp, Mpay, f);


def smTotalMassCalc(sm1Mass,sm2Mass,sm3Mass,sm4Mass):
    return sm1Mass + sm2Mass + sm3Mass + sm4Mass;

def s4MassCalc(smTotalMass, lmTotalMass, cmMass):
    Mpay = smTotalMass + lmTotalMass + cmMass;
    return StageMass(dVeod, Isp2, Mpay, f);

def IMLEOCalc(EOR,earthLaunch,LOR,moonArrival,moonDeparture,\
cmCrew,lmCrew,smFuel,lmFuel):
    cmMass = cmMassCalc(cmCrew);
    sm4Mass =  sm4MassCalc(cmMass,IspCalc(smFuel));
    lmMass = lmMassCalc(lmCrew);
    lm2Mass = lm2MassCalc(LOR,lmMass,IspCalc(lmFuel));
    lm1Mass = lm1MassCalc(LOR,lm2Mass,lmMass, IspCalc(lmFuel));
    lmTotalMass = lmTotalMassCalc(lm1Mass,lm2Mass,lmMass);
    sm3Mass = sm3MassCalc(LOR, sm4Mass,cmMass, IspCalc(smFuel) );
    sm2Mass = sm2MassCalc(LOR, sm3Mass, sm4Mass, \
    cmMass, IspCalc(smFuel));
    sm1Mass = sm1MassCalc(lmTotalMass,sm2Mass,sm3Mass,sm4Mass,\
    cmMass, IspCalc(smFuel));
    smTotalMass = smTotalMassCalc(sm1Mass,sm2Mass,sm3Mass,sm4Mass);
    s4Mass = s4MassCalc(smTotalMass, lmTotalMass, cmMass);
    IMLEO = s4Mass + smTotalMass + lmTotalMass + cmMass;
    return IMLEO;


def MissionRisk(EOR,earthLaunch,LOR,moonArrival,moonDeparture,\
cmCrew,lmCrew,smFuel,lmFuel):
    #start with 100%
    risk=1.0;
    #earth orbit
    if (EOR == no):
        risk = risk * .98;
    if (EOR == yes):
        risk = risk * .95;
        #second launch risk
        risk = risk * .95;
    if (earthLaunch == direct):
        risk = risk *.90;
    if (earthLaunch == orbit):
        risk = risk * .99;
    if (LOR==yes):
        risk = risk *.95;
    if (LOR==no):
        risk = risk * 1;
    if (moonArrival == orbit):
        #lunar orbit entry
        risk = risk * .99;
        #decend from lunar orbit
        risk = risk * .95;
    if (moonArrival == direct):
        #direct decending
        risk = risk * .90;
    if (moonDeparture == direct):
        risk = risk *.90
    if (moonDeparture == orbit):
        #direct ascent to earth
        risk = risk * .90;
    if (cmCrew-lmCrew == 0):
        #leaving CM unmanned
        risk = risk * .90;
    if (lmCrew == 1):
        risk = risk*.90;
    if (smFuel == cryogenic):
        # .98 for each burn
        if (LOR == yes):
            multiplier = 2.0;
        else:
            multiplier = 4.0;
        risk = risk * m.pow(0.95,multiplier);
    if (lmFuel == cryogenic):
        #two burns
        risk = risk * 0.95 * 0.95;
    #direct earth entry
    risk = risk * .95;
    return risk;



#Mass ratio: mass final / mass initial
def R(deltaV, specImp):
        x=(deltaV/(specImp*g))
        ratio = m.exp(-x)
        return ratio

#Mass ratio between total / payload
def Rt2p(deltaV, specImp,f):
        r = R(deltaV, specImp)
        return 1/(r - f + (f * r) )

#returns totoal initial mass of just this stage (minus payload)
def StageMass(deltaV, specImp, Mpay, f):
    totalPayRatio = Rt2p(deltaV, specImp, f);
    return (totalPayRatio * (Mpay)- Mpay);

def staged(deltaV, specImp,f,s):
        dV = deltaV/s
        r = R(dV, specImp)
        stage = (1/(r - f + (f * r) ))**s
        return stage

def IspCalc(fuelType):
    if (fuelType == cryogenic):
        return Isp2;
    if (fuelType == storable):
        return Isp1;
    else:
        return Isp1;
```

B

Supplemental Material for the Lunar Outpost Study

This appendix contains some additional information about the Lunar Outpost study presented in Chapter 5.

B.1 Alternate Views of the Lunar Outpost ADG

The complete, logical, and properties views of Parts A and B of the Lunar Outpost ADG are shown in this section.

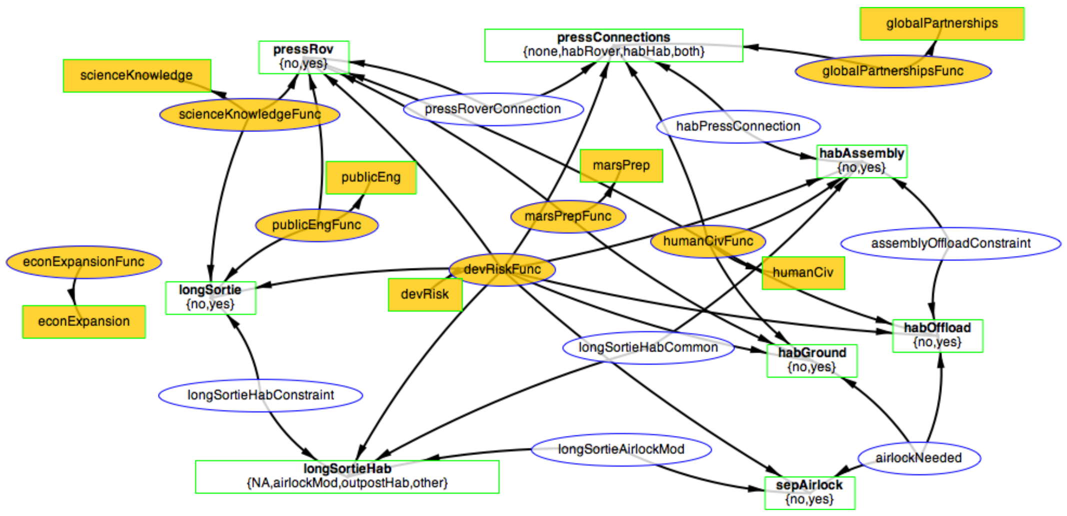


Figure B-1: Lunar Outpost ADG – Complete View of Part A.

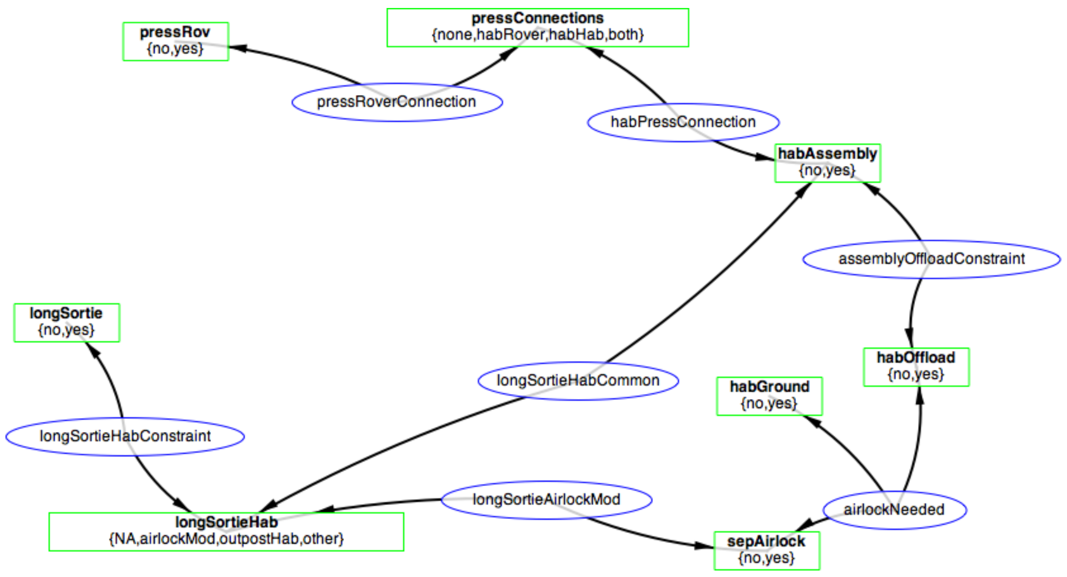


Figure B-2: Lunar Outpost ADG – Logical View of Part A.

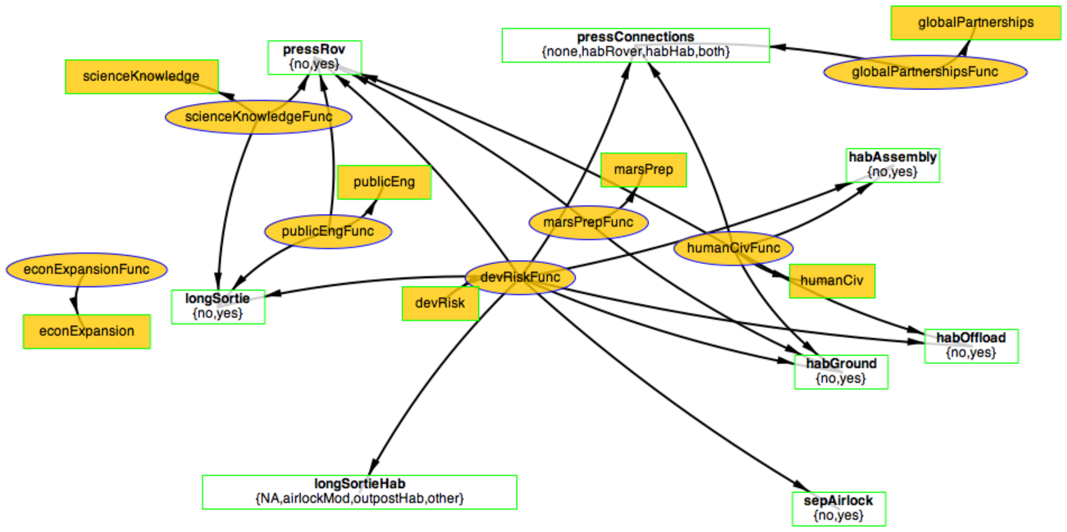


Figure B-3: Lunar Outpost ADG – Properties View of Part A.

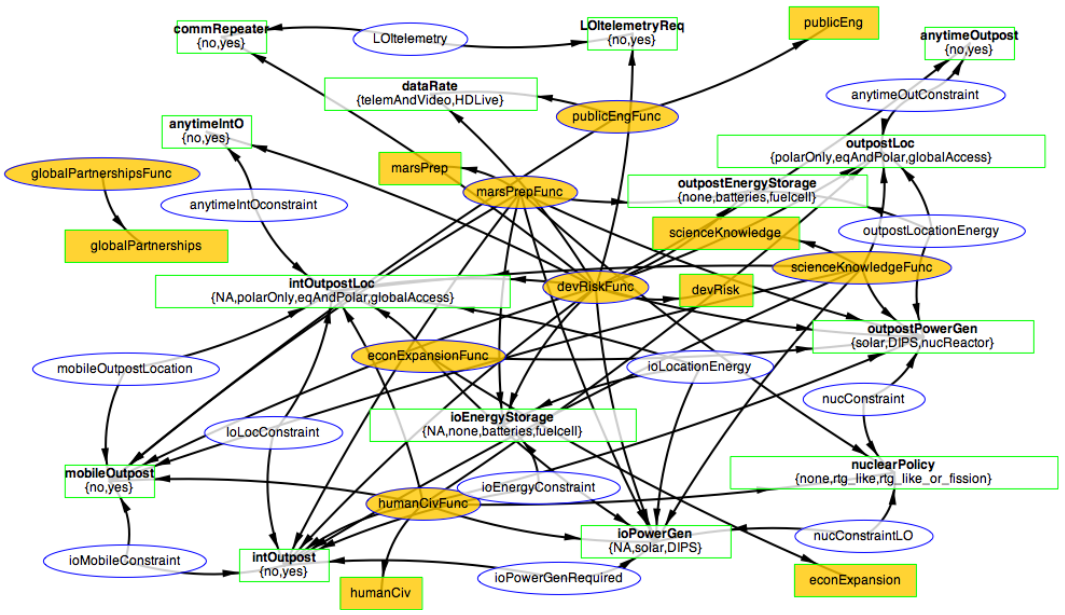


Figure B-4: Lunar Outpost ADG – Complete View of Part B.

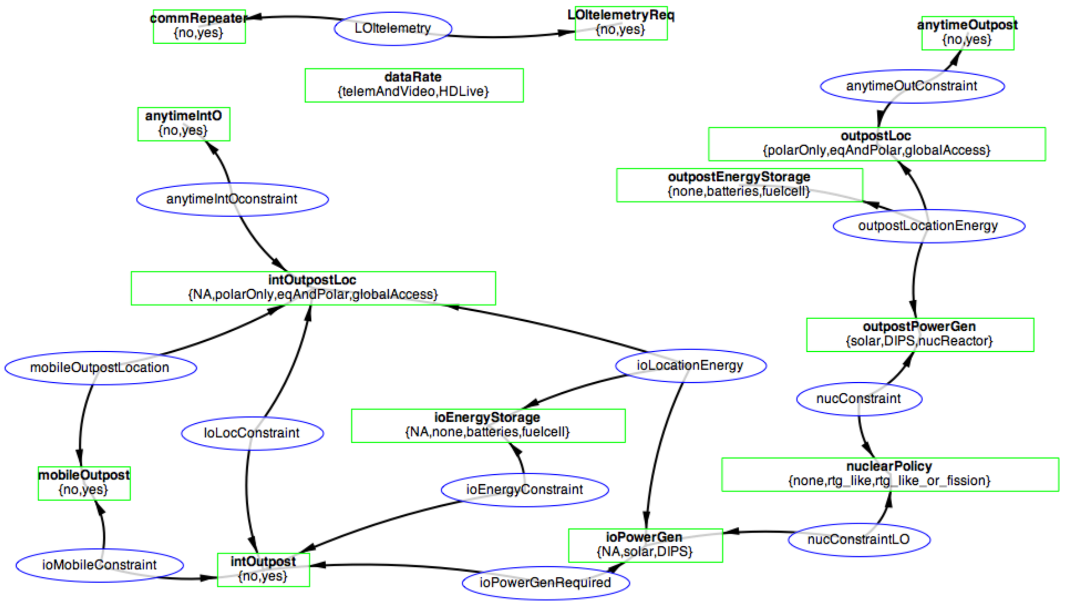


Figure B-5: Lunar Outpost ADG – Logical View of Part B.

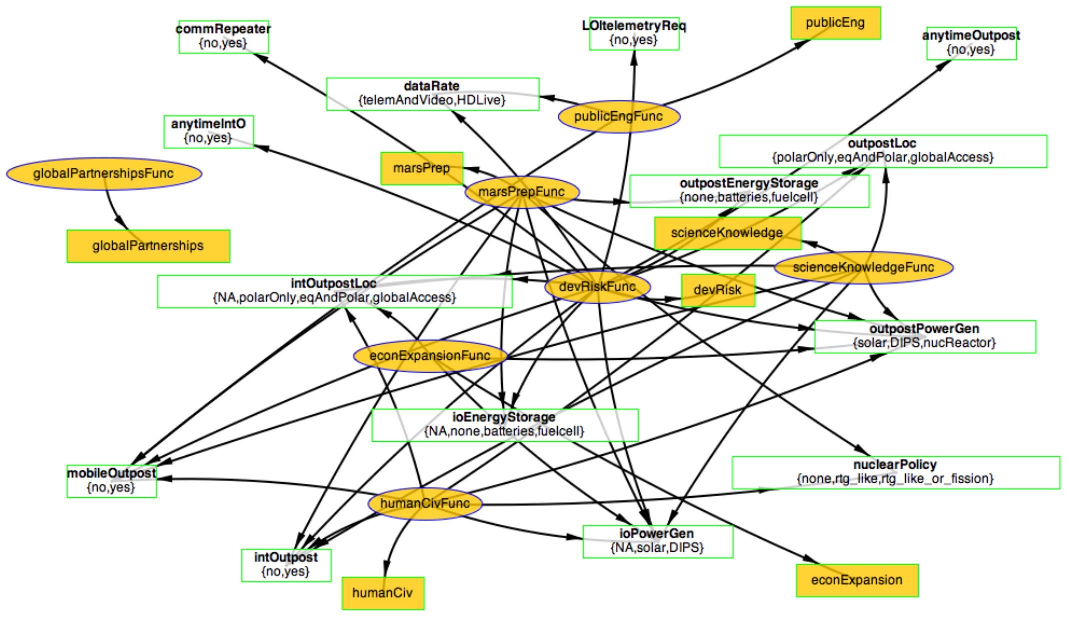

commRepeater {no,yes}
LOItelemetryReq {no,yes}
publicEng
anytimeOutpost {no,yes}
dataRate {telemAndVideo,HDLive}
publicEngFunc
anytimeIntO {no,yes}
outpostLoc {polarOnly,eqAndPolar,globalAccess}
globalPartnershipsFunc
marsPrep
marsPrepFunc
outpostEnergyStorage {none,batteries,fuelcell}
globalPartnerships
scienceKnowledge
scienceKnowledgeFunc
intOutpostLoc {NA,polarOnly,eqAndPolar,globalAccess}
devRiskFunc
devRisk
outpostPowerGen {solar,DIPS,nucReactor}
econExpansionFunc
ioEnergyStorage {NA,none,batteries,fuelcell}
nuclearPolicy {none,rtg_like,rtg_like_or_fission}
mobileOutpost {no,yes}
humanCivFunc
ioPowerGen {NA,solar,DIPS}
intOutpost {no,yes}
humanCiv
econExpansion

Figure B-6: Lunar Outpost ADG – Properties View of Part B.

Bibliography

[10905] 109th Congress of the United States of America. "National Aeronautics and Space Administration Authorization Act of 2005". H.R.3070, December 2005.

[Abr65] L. H. Abraham. *Space Technology Volume 1: Spacecraft Systems*. Number SP-65. NASA: Scientific and Technical Information Division, 1965.

[ah-04] *The American Heritage Dictionary of the English Language*. Houghton Mifflin Company, 4th edition, 2004.

[Ald04] Edward C. Aldridge, Jr. (Chairman of the Commision). A Journey to Inspire, Innovate, and Discover. Report of the President's Commission on Implementation of United States Space Exploration Policy, June 2004.

[Ale77] C. Alexander. *A Pattern Language*. Oxford University Press, New York, 1977.

[Ale79] C. Alexander. *The Timeless Way of Building*. Oxford University Press, New York, 1979.

[Alt80] Steven L. Alter. *Decision Support Systems: Current Practice and Continuing Challenges*. Addison-Wesley, 1980.

[BAH$^+$05] Gergana A. Bounova, Jaemyung Ahn, Wilfried Hofstetter, Paul Wooster, Rania Hassan, and Olivier L. de Weck. Selection and technology evaluation of moon/mars transportation architectures. In *AIAA Space 2005*, number AIAA-2005-6790, Long Beach, California, August 30 - September 1, 2005.

[BCE$^+$04] A. Bushman, D.M. Carpenter, T.S. Ellis, S.P. Gallagher, Hershcovitch, M.C. Hine, E.D. Johnson, S.C. Kane, M.R. Presley, A.H. Roach, S. Shaikh, M.P. Short, and M.A. Stawicki. The Martian Surface Reactor: An Advanced Nuclear Power Station for Manned Extraterrestrial Exploration. Nuclear Space Application Program Technical Report MIT-NSA-TR-003, Massachusetts Institute of Technology, Cambridge, MA, USA, December 2004.

[Ber05] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 3rd edition, 2005.

[BGS79] C. G. Brooks, J. M. Grimwood, and L. S. Swenson. *Chariots for Apollo: a history of manned lunar spacecraft.* NASA: Scientific and Technical Information Branch, 1979.

[BHH05] G.E.P. Box, J.S. Hunter, and W.G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley, 2005.

[BHW81] R.H. Bonczek, C.W. Holsapple, and A.B. Whinston. *Foundations of Decision Support Systems*. Academic Press, New York, 1981.

[BL85] Ronald J. Brachman and Hector J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, 1985.

[BLS98] C. Barnhart, F. Lu, and R. Shenoi. Integrated Airline Schedule Planning. *Operations Research in the Airline Industry*, 9:384–403, 1998.

[BM96] R. J. Bayardo and D. P. Miranker. A Complexity Analysis of Space-Bounded Learning Algorithms for the Constraint Satisfaction Problem. In *Proceedings of NCAI 1996*, 1996.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[Bus04] President George W. Bush. “The Vision for Space Exploration”. The White House, January 2004.

[Cam07] Bruce G. Cameron. Value network modeling. Master’s thesis, Massachusetts Institute of Technology, 2007.

[Cat06] Sandro N. Catanzaro. Multi-stakeholder quantitative analysis of sustainability for value delivery systems. Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2006.

[CCC06] Bruce G. Cameron, Sandro Catanzaro, and Edward F. Crawley. Value based architecture selection. In *57th International Astronautical Congress*, 2006.

[CdWE$^+$04] Edward Crawley, Olivier de Weck, Steven Eppinger, Christopher Magee, Joel Moses, Warren Seering, Joel Schindall, David Wallace, and Daniel Whitney (chair). The Influence of Architecture in Systems Engineering. Engineering Systems Monograph, Massachusetts Institute of Technology, Cambridge, MA, 2004.

[Cho96] Vladimir A. Chobotov, editor. *Orbital Mechanics*. AIAA, 2nd edition, 1996.

[CN06] Doug Cooke and NASA Exploration Systems Mission Directorate. Exploration Strategy and Architecture. Presentation at Implementing the Vision, 2nd Space Exploration Conference, December 6 2006.

[CO95] Zvi Covaliu and Robert M. Oliver. Representation and solution of decision problems using sequential decision diagrams. *Management Science*, 41(12):1860–1881, 1995.

[Cra07] E. F. Crawley. ESD.34 Systems Architecting – lecture notes. MIT Engineering Systems Division, IAP 2007.

[DAE+05] C.L. Dym, A.M. Agogino, O. Eris, D.D. Frey, and L.J. Leifer. Engineering Design Thinking, Teaching, and Learning. *Journal of Engineering Education*, 94(1):103–120, 2005.

[de 90] Richard de Neufville. *Applied Systems Analysis: Engineering Planning and Technology Management*. McGraw-Hill, 1990.

[Dec03] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[Dec04] Rina Dechter. AND/OR search spaces for graphical models. Technical report, UCLA ICS, March 2004.

[Dep07] Department of Defense. DoD Architecture Framework Version 1.5, April 2007.

[dic08] "explicit.". Dictionary.com Unabridged (v 1.1). Random House, Inc. `http://dictionary.reference.com/browse/explicit`, 05 Jan 2008.

[dLMS03] Simon de Givry, Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Solving Max-SAT as weighted CSP. In *CP2003: Principles and Practice of Constraint Programming : (Kinsale, 29 September - 3 October 2003)*, 2003.

[DM07] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, 2007.

[Dor02] Dov Dori. *Object-Process Methodology*. Springer, Aug 2002.

[DP89] Rina Dechter and Judea Pearl. Tree clustering for constraint networks (research note). *Artif. Intell.*, 38(3):353–366, 1989.

[Dra05] Draper - MIT Team. Concept Exploration and Refinement Study. Final Report, September 15 2005.

[dsm] Design structure matrix (DSM). Website: http://www.dsmweb.org.

[EMB+78] Ivan D. Ertel, Mary Louise Morse, Jean Kernahan Bays, Courtney G. Brooks, and Roland W. Newkirk. *The Apollo Spacecraft: A Chronology*, volume 1-4. NASA SP-4009, 1969-1978.

[EWSG94] S.D. Eppinger, D.E. Whitney, R.P. Smith, and D.A. Gebala. A model-based method for organizing tasks in product development. *Research in Engineering Design*, 6(1):1–13, 1994.

[Exp05] Exploration Systems Architecture Study Team. Final report. NASA, November 2005.

[Far67] R. W. Farquhar. Lunar Communications with Libration-Point Satellites. *Journal of Spacecraft and Rockets*, 4:1383–1384, 1967.

[FHK93] L. Fang, Keith W. Hipel, and D. M. Kilgour. *Interactive Decision Making: The Graph Model for Conflict Resolution.* Wiley, 1993.

[Fra01] Emilio Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2001.

[GSS$^+$06] W.G. Griswold, M. Shonle, K. Sullivan, Y. Song, N. Tewari, Y. Cai, and H. Rajan. Modular software design with crosscutting interfaces. *IEEE Software*, 23(1):51–60, 2006.

[Han95] James R. Hansen. *Spaceflight Revolution: NASA Langley Research Center From Sputnik to Apollo.* The NASA History Series. NASA SP-4308, Washington, D.C., 1995.

[Han99] James R. Hansen. Enchanted Rendezvous. NASA Monographs in Aerospace History #4, January 25 1999.

[Han05] Sven Ove Hansson. Decision theory a brief introduction. Technical report, Royal Institute of Technology (KTH) – Department of Philosophy and the History of Technology, Stockholm, 2005.

[Har99] James Harford. *Korolev: How One Man Masterminded the Soviet Drive to Beat America to the Moon.* Wiley, 1999.

[Hil04] Tom Hill. “Decision Point”. *The Space Review*, November 9, 2004.

[HKG01] Stephen J. Hoch, Howard C. Kunreuther, and Robert E. Gunther. *Wharton on Making Decisions.* Wiley, 2001.

[HKLP97] Keith W. Hipel, D. M. Kilgour, F. Liping, and X. Peng. The Decision Support System GMCR in Environmental Conflict Management. *Applied Mathematics and Computation*, 83(2):117–152, 1997.

[HKZ07] Yi Huang, Benjamin H. Y. Koo, and Li Zheng. Modeling reconfigurable information systems by using an executable algebraic framework. In *Proceedings of IEEE International Conference on Industrial Engineering and Engineering Management (IEEM2007)*, December 2007.

[HM84] Ronald A. Howard and James E. Matheson. Influence diagrams. In Ronald A. Howard and James E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, Menlo Park, California, 1984.

[HM05] Ronald A. Howard and James E. Matheson. Influence diagrams. *Decision Analysis*, 2(3):127–143, September 2005.

[Hof04] Wilfried K. Hofstetter. Extensible Modular Landing Systems for Human Moon and Mars Exploration. Diplomarbeit for Technical University of Munich, Germany, December 2004.

[Hof05] Robert R. Hoffman. Decision Making: human-centered computing. *IEEE Intelligent Systems*, 20(4):76–83, July-August 2005.

[Hou61a] J. C. Houbolt. Manned lunar-landing through use of lunar-orbit rendezvous. Technical Report NASA-TM-74736, NASA, 1961.

[Hou61b] John C. Houbolt. Problems and Potentialities of Space Rendezvous. *Acta Astronautica*, 1961.

[HSWC07] Wilfried K. Hofstetter, Timothy A. Sutherland, Paul D. Wooster, and Edward F. Crawley. The intermediate outpost - an alternate concept for human lunar exploration. In *Space 2007*, September 2007.

[HWC07] Wilfried K. Hofstetter, Paul D. Wooster, and Edward F. Crawley. Analysis of Human Lunar Outpost Strategies and Architectures. In *Proceedings of AIAA Space 2007*, September 2007.

[HWNC05] W. Hofstetter, P. Wooster, W. Nadir, and E. F. Crawley. Affordable human moon and mars exploration through hardware commonality. In *AIAA Space 2005*, number AIAA-2005-6757, Long Beach, California, August 30 - September 1, 2005.

[HWSC06] Wilfried K. Hofstetter, Paul D. Wooster, Timothy A. Sutherland, and Edward F. Crawley. Architecture and Design Options for NASA's Lunar Surface Access Module (LSAM). In *57th International Astronautical Congress*, 2006.

[KDLD05] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):165–193, 2005.

[Ken61] John. F. Kennedy. Special message to the congress on urgent national needs. Speech, delivered in person before a joint session of Congress, May 25 1961.

[Ker05] Thomas W. Kerslake. Electric Power System Technology Options for Lunar Surface Missions. Technical Report TM—2005-213629, NASA, Glenn Research Center, Cleveland, Ohio, April 2005.

[Kin05] Matti J. Kinnunen. "Interview with Dr. John Houbolt". electronic, April 8 2005.

[Kir93] Craig W. Kirkwood. An algebraic approach to formulating and solving large models for sequential decisions under uncertainty. *Management Science*, 39(7):900–913, 1993.

[Kle99] G. A. Klein. *Sources of Power: How People Make Decisions.* MIT Press, 1999.

[Kle05] Howard Neil Kleinwacks. A mars-back approach to lunar surface operations. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2005.

[KLM$^+$97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[Koo05] Benjamin H. Y. Koo. *A Meta-language for Systems Architecting.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2005.

[KSC07a] Benjamin H. Y. Koo, Willard L. Simmons, and Edward F. Crawley. Algebra of Systems: a meta-language for model synthesis and evaluation. (unpublished, manuscript submitted for publication), January 2007.

[KSC07b] Benjamin H. Y. Koo, Willard L. Simmons, and Edward F. Crawley. Algebra of systems: an executable framework for model synthesis and evaluation. In *Proceedings of the 2007 International Conference on Systems Engineering and Modeling*, 2007.

[KSC07c] Benjamin H. Y. Koo, Willard L. Simmons, and Edward F. Crawley. A valuation technology for product development options using an executable meta-modeling language. In Geilson Loureiro and Richard Curran, editors, *Complex Systems Concurrent Engineering: Collaboration, Technology Innovation and Sustainability*, pages 107–115. Springer, 2007.

[LCCR06] Geilson Loureiro, Edward F. Crawley, Sandro Catanzaro, and Eric Rebentisch. From value to architecture - ranking the objectives of space exploration. In *Proceedings of the 57th International Astronautical Congress*, Valencia, Spain, 2006.

[Lev00] Nancy G. Leveson. Intent specifications: An approach to building human-centered specifications. *IEEE Transactions on Software Engineering*, 26(1):15–35, January 2000.

[LHR99] Nancy G. Leveson, Mats Heimdahl, and Jon Damon Reese. Designing specification languages for process control systems: Lessons learned and steps to the future. In *SIGSOFT FOSE '99 (Foundations of Software Engineering)*, Toulouse, France, September 1999.

[Mae06] John Maeda. *The Laws of Simplicity.* MIT Press, 2006.

[Mat07] Robert Eugeniu Mateescu. *AND/OR Search Spaces for Graphical Models.* PhD thesis, University of California, Irvine, June 2007.

[MC04] C. Murray and C. B. Cox. *Apollo.* South Mountain Books, 2004.

[Mon82] G.E. Monahan. A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science*, 28(1):1–16, 1982.

[MR02] Mark W. Maier and Eberhardt Rechtin. *The Art of Systems Architecting.* CRC Press, 2nd edition, 2002.

[NAS06a] NASA. Lunar Exploration Objectives. electronic distribution through NASA.gov, December 2006.

[NAS06b] NASA. "why the moon?", `http://www.nasa.gov/pdf/163561main_why_moon2.pdf`. NASA Public Relations Poster, December 4 2006.

[NAS06c] NASA Headquarters. NASA Selects Orion Crew Exploration Vehicle Prime Contractor. Press Release 06-305, August 31 2006.

[NAS06d] NASA Lunar Architecture Team (LAT). Briefing on Lunar Exlporation Strategy and Architecture. In *Proceedings of 2nd Space Exploration Conference*, Houston, TX, December 2006.

[NAS07] NASA Exploration Systems Mission Directorate. Lunar Architecture Update. Presentation at AIAA Space 2007, September 20 2007.

[OMG03] OMG. *Unified Modeling Language (UML), version 1.5*, volume 1. Object Management Group, 2003.

[Orl04] Richard W. Orloff. *APOLLO BY THE NUMBERS: A Statistical Reference SP-2000-4029.* NASA History Division, Washington, D.C., September 2004.

[Par97] Vilfredo Pareto. Cours d'Economique Politique. *Lausanne & Paris*, 1897.

[Par69] Vilfredo Pareto. *Manual of Political Economy.* Augustus M Kelley Pubs, reprint edition, May 1969.

[PB95] G. Pahl and W. Beitz. *Engineering Design: A Systematic Approach.* Springer, 2nd edition, 1995.

[Pet62] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, University of Bonn, 1962.

[Pet81] J. L. Peterson. *Petri-Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliff, N.J, 1981.

[Pow02] Daniel J. Power. *Decision Support Systems: Concepts and Resources for Managers.* Quorum Books, 2002.

[PR02] S. Pedroni and N. Rapping. *Jython Essentials.* O'Reilly, Sebastopol, CA, March 2002.

[Puc06] Daniela Pucci de Farias. 2.193 decision-making in large scale systems – lecture notes. MIT Department of Mechanical Engineering, Spring 2006.

[PW00] Panos Y. Papalambros and Douglass J. Wilde. *Principles of Optimal Design: Modeling and Computation.* Cambridge University Press, 2nd edition, 2000.

[Rab89] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[Rai68] Howard Raiffa. *Decision Analysis: Introductory Lectures on Choices Under Uncertainty.* Addison-Wesley, 1968.

[RCL$^+$05] Eric S. Rebentisch, Edward F. Crawley, Geilson Loureiro, John Q. Dickmann, and Sandro N. Catanzaro. Using stakeholder value analysis to build exploration sustainability. In *Proceedings of 1st Space Exploration Conference*, Orlando, Florida, 2005. AIAA.

[Rec91] Eberhardt Rechtin. *Systems Architecting: Creating and Building Complex Systems.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[Rit06] T. Ritchey. Problem structuring using computer-aided morphological analysis. *Journal of the Operational Research Society*, 57:792–801, 2006.

[RJ86] Lawrence R. Rabiner and B. Juang. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

[RN02] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2nd edition, 2002.

[Roa96] Curtis Roads. *The Computer Music Tutorial.* MIT Press, 1996.

[Roz07] Ziv Rozenblum. Object-Process Networks & Object-Process Diagrams – Implementation Issues for Oil Exploration Systems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 2007.

[Sch07] Jeffry G. Schreiber. Status of the NASA Stirling Radioisotope Project. Technical Report TM—2007-214804, NASA, Glenn Research Center, Cleveland, Ohio, May 2007.

[SdW07] Matthew R. Silver and Olivier L. de Weck. Time-Expanded Decision Networks: A framework for designing evolvable complex systems. *Systems Engineering*, 10(2):167–188, 2007.

[Sea96] Robert C. Seamans. *Aiming At Targets: The Autobiography Of Robert C. Seamans, Jr.* NASA, 1996.

[Sea05] Robert C. Seamans, Jr. *Project Apollo: The Tough Decisions.* Monographs in Aerospace History Number 37. NASA SP-2005-4537, Washington, D.C., 2005.

[SFV95] Thomas Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proceedings of IJCAI95*, Montréal, Canada, 1995.

[Sid03] Asif A. Siddiqi. *The Soviet Space Race with Apollo.* University of Florida, 2003.

[Sim60] Herbert A. Simon. *The New Science of Management Decision.* Harper and Brothers, New York, 1960.

[Sim77] Herbert A. Simon. *The New Science of Management Decision.* Prentice-Hall, Englewood Cliffs, N.J., 3rd edition, 1977.

[SKC05a] Willard L. Simmons, Benjamin H. Y. Koo, and Edward F. Crawley. Architecture generation for Moon-Mars exploration using an executable meta-language. In *Proceedings of AIAA Space 2005, 30 August - 1 September, Long Beach, CA*, 2005.

[SKC05b] Willard L. Simmons, Benjamin H. Y. Koo, and Edward F. Crawley. Space systems architecting using meta-languages. In *56th International Astronautical Congress*, 2005.

[SKC06] Willard L. Simmons, Benjamin H. Y. Koo, and Edward F. Crawley. A computational method for mapping the decision space of the lunar exploration program. In *57th International Astronautical Congress*, 2006.

[SKC07] Willard L. Simmons, Benjamin H. Y. Koo, and Edward F. Crawley. Decision-making for systems architecting using meta-languages. In *Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC2007)*, Montréal, Canada, October 2007.

[SKK05] Willard L. Simmons, Matti Kinnunen, and Benjamin H. Y. Koo. "Interview with Robert C. Seamans", 1 April 2005.

[SPL07] Felipe Simon, Gustavo Pinheiro, and Geilson Loureiro. Towards Automatic Systems Architecting. In Geilson Loureiro and Richard Curran, editors, *Complex Systems Concurrent Engineering: Collaboration, Technology Innovation and Sustainability*, pages 117–130. Springer, 2007.

[Ste65] D. Steward. Partitioning and tearing systems of equations. *SIAM Numerical Analysis*, B, 2(2):345–365, 1965.

[Ste81] Donald V. Steward. The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management*, 28:71–74, 1981.

[Sys06] SysML Partners. OMG Systems Modeling Language (OMG SysML^{TM}) Specification, April 2006.

[TA98] Efraim Turban and Jay E. Aronson. *Decision Support Systems and Intelligent Systems*. Prentice Hall, Upper Saddle River, New Jersey, 5th edition, 1998.

[TAL05] Efraim Turban, Jay E. Aronson, and Ting-Peng Liang. *Decision Support Systems and Intelligent Systems*. Prentice Hall, Upper Saddle River, New Jersey, 7th edition, 2005.

[Tan06] Victor Tang. *Corporate Decision Analysis: An Engineering Approach*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2006.

[TBB$^+$05] Christine Taylor, David Broniatowski, Ryan Boas, Matt Silver, Edward Crawley, Olivier de Weck, and Jeffrey Hoffman. Paradigm Shift in Design for NASA's Space Exploration Initiative: Results from MIT's Spring 2004 Study. In *Proceedings of the 1st Space Exploration Conference: Continuing the Voyage of Discovery*, number AIAA 2005-2766, Orlando, Florida, 30 January - 1 February 2005. AIAA.

[Tea07] Team of 14 Space Agencies: ASI (Italy); BNSC (UK); CNES (France); CNSA (China); CSA (Canada); CSIRO (Australia); DLR (Germany); ESA (European Space Agency); ISRO (India); JAXA (Japan); KARI (Republic of Korea); NASA (USA); NSAU (Ukraine); and, Roscosmos

(Russia). The Global Exploration Strategy: The Framework for Coordination. published at the 3rd ASI/ESA International Co-operation for Sustainable Space Exploration Workshop, 29 May to 01 June 2007.

[vB53] W. von Braun. *Conquest of the Moon.* Viking Press, 1953.

[WDR93] D. Weaver, M. B. Duke, and B. Roberts. Mars exploration strategies: A reference design mission. In *IAF 93-Q.1.383*. IAF, 1993.

[Wer99] James Richard Wertz. *Space Mission Analysis and Design.* Springer, 3rd edition, 1999.

[WH00] Christopher D. Wickens and Justin G. Hollands. *Engineering Psychology and Human Performance.* Prentice Hall, 3rd edition, 2000.

[WH03] Annalisa L. Weigel and Daniel E. Hastings. Interaction of policy choices and technical requirements for a space transportation infrastructure. *Acta Astronautica*, (7):551–562, April 2003.

[WHC05a] P. Wooster, W. Hofstetter, and E. F. Crawley. Crew exploration vehicle destination for human lunar exploration: The lunar surface. In *AIAA Space 2005*, number AIAA-2005-6626, Long Beach, California, August 30 - September 1, 2005.

[WHC05b] P. Wooster, W. Hofstetter, and E. F. Crawley. Crew exploration vehicle destination for human lunar exploration: The lunar surface. In *AIAA Space 2005*, number AIAA-2005-6626, Long Beach, California, August 30 - September 1, 2005.

[WHI07] Sean Walker, Keith W. Hipel, and Takehiro Inohara. Strategic Analysis of the Kyoto Protocol. In *Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC2007)*, 2007.

[Wie95] Wiesner Committee. "Report to the President-Elect of the Ad Hoc Committee on Space", 10 January 1961. In *Exploring the Unknown: Selected Documents in the History of the U.S. Civil Space Program Volume I: Organizing for Exploration.* NASA SP-4407, 1995.

[Wik07] Wikipedia. Hidden markov model — wikipedia, the free encyclopedia, 2007. [Online; accessed 29-October-2007].

[Wil89] John Noble Wilford. "Russians Finally Admit They Lost Race to Moon". New York Times, December 18 1989.

[Woo07] Paul D. Wooster. Practical Approaches to Cost Effective Human Moon and Mars Exploration. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[WW03] Karen Willcox and Sean Wakayama. Simultaneous optimization of a multiple-aircraft family. *Journal of Aircraft*, 40(4):616–622, July–August 2003.

[XKZ07] Lan Xiao, Benjamin H. Y. Koo, and Li Zheng. Achieve flexibility in business process modeling using an algebraic language. unpublished: submitted to the Journal of Tsinghua Science and Technology, December 2007.

[ZBG91] Robert M. Zubrin, David A. Baker, and Owen Gwynne. Mars direct: A simple robust and cost effective architecture for the space exploration initiative. In *Aerospace Sciences Meeting, 29th, Reno, NV*, number AIAA-91-0328 or AIAA-91-0328, page 28, Jan. 7-10, 1991.

[Zub97] R. Zubrin. *The Case for Mars*. Touchstone, 1997.

[Zwi66] Fritz Zwicky. *Discovery, Invention, Research through the morphological approach*. Macmillan, 1966.