8-2016

Supporting Space Systems Design via Systems Dependency Analysis Methodology

Cesare Guariniello
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Cesare Guariniello

Entitled

SUPPORTING SPACE SYSTEMS DESIGN VIA SYSTEMS DEPENDENCY ANALYSIS METHODOLOGY

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Daniel A. DeLaurentis
Chair

James M. Longuski

Kathleen C. Howell

William A. Crossley

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Daniel A. DeLaurentis

Approved by: Weinong Wayne Chen 7/25/2016

Head of the Departmental Graduate Program Date

SUPPORTING SPACE SYSTEMS DESIGN VIA SYSTEMS DEPENDENCY ANALYSIS METHODOLOGY

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Cesare Guariniello

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2016

Purdue University

West Lafayette, Indiana

To my mom and dad, who gave me life and taught me the meaning of love

To my sister Maria, whom I have always been proud to grow up with

To Anna Laura, who shared all of the tough moments during my research

ACKNOWLEDGMENTS

First of all, I would like to thank my research advisor and chair of my Ph.D. committee, Dr. Daniel DeLaurentis. Without his help, I would have not been able to conduct this research and to get closer to the realization of my wishes. He patiently advised and guided me through all the steps in the process, giving me chances to learn and produce to the best of my capabilities. I would also like to recognize the other members of my Ph.D. committee: Dr. William Crossley, Dr. Kathleen Howell, and Dr. James Longuski. I am grateful for their support and encouragement during my years at Purdue University. I appreciate the involvement of the United States Department of Defense and of the Systems Engineering Research Center (SERC), that through Dr. DeLaurentis provided financial support to my research and my residence in the United States, and the assistance of Prof. Daniel Dumbacher, whose expertise in the space industry and support of the application of my research to space projects provided a substantial help.

During my work within the SERC project, I was proud to work with inspiring and supportive faculty and colleagues. I am grateful to Dr. Karen Marais for her useful advice, and to Seung-Yeob Han and Ankur Mour for sharing with me the first few years of this project. I am deeply indebted to Zhemei Fang, Dr. Payuna Uday and Dr. Navindran Davendralingam. They shared hours and hours of work and fun with me, and they supported me inside and outside academia. Their friendship and encouragement never failed to help me overcome the toughest moments of this research effort.

My appreciation goes also to a long list of friends and colleagues, who give plenty of wealth to my life, and without whom it would have been much harder to achieve outstanding results. They never ceased to believe in me, and even if I do not name all of them personally, I am deeply grateful to share the paths of life with them.

All of my friends in Italy, especially Danilo Defant, the members of the Choir of the Diocese of Rome, the faculty and students of the School of Aerospace Engineering in Rome, and all those who kept looking for my advice and my company even when I moved five thousand miles away. The past and current members of the System-of-Systems research group at Purdue University. My friends and colleagues in the School of Aeronautics and Astronautics, in the department of Earth, Atmospheric and Planetary Science, and in my Russian classes at Purdue University. My friends and colleagues in the honor society for Aerospace Engineering Sigma Gamma Tau. My house mates, Keenan Shimko, who taught me so much in my first years in the United States, Shawn Merrill, who shared one year with us, and Dr. Marat Kulakhmetov, that helped me prove that a house with two aerospace engineers can never be boring. My friends and teammates in Purdue Fencing Club, with whom I had the honor to represent Purdue University in collegiate events. My gratitude goes especially to Dr. Mark J.T. Smith, who has been an outstanding coach and mentor, and to Stephen and Ashley Wien, Anna-Elodie Kerlo, and Jacob Miller, whose friendship made me always feel at home and loved, and gave me strength to keep pushing forward. I am grateful to Nicoletta Fala, who has been the first to let me try the joy of flying in a single-engine General Aviation aircraft, and who is sharing much needed relaxing training sessions with me.

My deepest appreciation and love to my father Giovanni Michele, my mother Angela and my sister Maria, who backed up my decision to leave in pursue of my own path and accepted the long distance out of love and with much patience. They always supported me, and rejoiced for all my achievements. I am also grateful to Anna Laura, who for years shared the burden and the difficulties of long distance, but has always been supportive and ready to listen to my grievances and encourage me to follow my dreams.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

SYMBOLS

E	set of edges in a graph. Each edge is an ordered pair of nodes (i, j)
$G = (N, E)$	graph (network) comprised of a set of nodes N and a set of edges (ordered pairs of nodes) E
N	set of nodes in a graph
O_i	operability of node i
O_i^C	term of operability of node i depending on Criticality of Dependency parameters
O_i^S	term of operability of node i depending on Strength of Dependency parameters
P_i	punctuality of node i
$Res_{\mathcal{A}^*}$	resilience of architecture $\mathcal{A}$ when flexibility results in alternate architecture $\mathcal{A}^*$
$Res^N_{\mathcal{A}^*}$	resilience of architecture $\mathcal{A}$ against failures in node N when flexibility results in alternate architecture $\mathcal{A}^*$
$Rob_{\mathcal{A}}$	robustness of architecture $\mathcal{A}$
$Rob^N_{\mathcal{A}}$	robustness of architecture $\mathcal{A}$ to failures in node N
t^j_B	beginning time of development of j
t^j_C	completion time of development of j
t^j_D	total development time of j
${}^i t^j_B$	beginning time of development of j based on its dependency on i
${}^i t^j_C$	completion time of development of j based on its dependency on i
W_{ij}	average operability of feeders of node j, excluding node i
α_{ij}	parameter for Strength of Dependency between i and j

β_{ij}	parameter for Criticality of Dependency between i and j
γ_{ij}	parameter for Impact of Dependency between i and j
Δv	delta-v: impulse for orbital maneuvers

ABBREVIATIONS

ABM	Agent Based Model
BN	Bayesian Networks
CM	Command Module
COD	Criticality of Dependency
CPM	Critical Path Method
DAF	Discrete Agent Framework
DODAF	Department of Defense Analytical Framework
DoE	Design of Experiments
DSM	Design Structure Matrix
EDL	Entry, Descent and Landing
ESA	European Space Agency
ETS-VII	Engineering Test Satellite - VII
FDNA	Functional Dependency Network Analysis
FFD	Full Factorial Design
FMEA	Failure Mode and Effect Analysis
FTA	Fault Tree Analysis
GEO	Geostationary Earth Orbit
GINA	Generalized Information Network Analysis
GNC	Guidance, Navigation, and Control
INCOSE	International Council on Systems Engineering
IOD	Impact of Dependency
ISRU	In-Situ Resource Utilization
IT	Information Technology
JAXA	Japanese Aerospace Exploration Agency
LCS	Littoral Combat Ship

LEO	Low Earth Orbit
LM	Lunar Module
MCC	Mission Control Center
MDM	Multiple Domain Matrix
MEO	Medium Earth Orbit
MRO	Mars Reconnaissance Orbiter
MSL	Mars Science Laboratory
NASA	National Aeronautics and Space Administration
NASDA	Japanese Agency for Space Development
NWS	Naval Warfare Scenario
ORU	Orbital Replacement Unit
PDF	Probability Density Function
PERT	Project Evaluation Research Technique
RMS	Root Mean Square
ROPE	Resources, Operations, Policies, Economics
RPD	Robust Parameter Design
RPO	Robust Portfolio Optimization
RSM	Response Surface Methodology
SDDA	Systems Developmental Dependency Analysis
SE	Self-Effectiveness
SLS	Space Launch System
SOD	Strength of Dependency
SODA	Systems Operational Dependency Analysis
SoS(s)	System(s)-of-Systems
SysML	Systems Modeling Language
TRL	Technology Readiness Level

GLOSSARY

Architecture	The organization of systems and capabilities in dependency networks, accounting for stakeholders decisions, policies, economics, technology requirements, and objectives.
Bottom-up	Approach based on the initial specification of the individual base elements in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy may result in a tangle of elements and subsystems, developed in isolation and subject to local optimization as opposed to meeting a global purpose.
Lead time	Latency between the initiation and execution (or the completion) of a process. Here used as latency between the initiation of a new process and the completion of a preceding process.
Resilience	Capability of a system to react to failures and degradation in order to recover operability, at least partially.
Robustness	Capability of a system to keep high performance following disruptions.
Stakeholder	Person or entity with an interest or concern in something, especially a business.

Top-down — Approach based on the initial formulation of an overview of the systems, where subsystems are specified but not detailed. Following the big picture, the system is broken down into smaller subsystems which are then refined in greater detail, until the entire specification reaches the base elements. This strategy may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model.

ABSTRACT

Guariniello, Cesare PhD, Purdue University, August 2016. Supporting Space Systems Design via Systems Dependency Analysis Methodology. Major Professor: Daniel DeLaurentis.

The increasing size and complexity of space systems and space missions pose severe challenges to space systems engineers. When complex systems and Systems-of-Systems are involved, the behavior of the whole entity is not only due to that of the individual systems involved but also to the interactions and dependencies between the systems. Dependencies can be varied and complex, and designers usually do not perform analysis of the impact of dependencies at the level of complex systems, or this analysis involves excessive computational cost, or occurs at a later stage of the design process, after designers have already set detailed requirements, following a bottom-up approach. While classical systems engineering attempts to integrate the perspectives involved across the variety of engineering disciplines and the objectives of multiple stakeholders, there is still a need for more effective tools and methods capable to identify, analyze and quantify properties of the complex system as a whole and to model explicitly the effect of some of the features that characterize complex systems.

This research describes the development and usage of Systems Operational Dependency Analysis and Systems Developmental Dependency Analysis, two methods based on parametric models of the behavior of complex systems, one in the operational domain and one in the developmental domain. The parameters of the developed models have intuitive meaning, are usable with subjective and quantitative data alike, and give redirect insight into the causes of observed, and possibly emergent, behavior. The approach proposed in this dissertation combines models of one-to-one dependencies among systems and between systems and capabilities, to analyze and evaluate

the impact of failures or delays on the outcome of the whole complex system. The analysis accounts for cascading effects, partial operational failures, multiple failures or delays, and partial developmental dependencies. The user of these methods can assess the behavior of each system based on its internal status and on the topology of its dependencies on systems connected to it. Designers and decision makers can therefore quickly analyze and explore the behavior of complex systems and evaluate different architectures under various working conditions.

The methods support educated decision making both in the design and in the update process of systems architecture, reducing the need to execute extensive simulations. In particular, in the phase of concept generation and selection, the information given by the methods can be used to identify promising architectures to be further tested and improved, while discarding architectures that do not show the required level of global features. The methods, when used in conjunction with appropriate metrics, also allow for improved reliability and risk analysis, as well as for automatic scheduling and re-scheduling based on the features of the dependencies and on the accepted level of risk.

This dissertation illustrates the use of the two methods in sample aerospace applications, both in the operational and in the developmental domain. The applications show how to use the developed methodology to evaluate the impact of failures, assess the criticality of systems, quantify metrics of interest, quantify the impact of delays, support informed decision making when scheduling the development of systems and evaluate the achievement of partial capabilities.

A larger, well-framed case study illustrates how the Systems Operational Dependency Analysis method and the Systems Developmental Dependency Analysis method can support analysis and decision making, at the mid and high level, in the design process of architectures for the exploration of Mars. The case study also shows how the methods do not replace the classical systems engineering methodologies, but support and improve them.

1. INTRODUCTION

1.1 Challenges in the 21^{st} century

"The more we realize our minuteness and our impotence in the face of cosmic forces, the more astonishing becomes what human beings have achieved."

Bertrand Arthur William Russell
If we are to survive this dark time
The New York Times Magazine, September 3rd, 1950

In 1964, as described by Eugene Francis "Gene" Kranz in his autobiography *Failure is not an option* [1], the National Aeronautics and Space Administration (NASA) changed the standard for the description of its space activities in all official documents. In the Mercury program, which inherited from aircraft flight testing, the word *flight* referred to both spacecraft and launcher events. Instead, during the Gemini and Apollo programs, the term *mission* was to be used when referring to the whole set of launcher, capsule, ground control centers, astronauts, space flight schedule, *et cetera*. This decision was more than a simple change in phrasing, because it reflected an entirely new concept: while in the Mercury program the spacecraft and launcher were considered to be a single system, in Gemini and Apollo programs NASA recognized the need to have a different perspective when dealing with multiple spacecraft and complex systems. Despite this change, however, the approach to design and architecture was still very independent for each of the systems involved; often different contractors built each of these systems. This procedure gave rise to hundreds of requirements for integration, but the resources were not enough to address the interest in architecture flexibility, that is the property of systems to be used in different configurations in an architecture and to provide additional functions and capabilities

beyond their main function. Furthermore, no thorough analysis of the effect of disruption of a system on the entire mission was performed. Every system was considered highly critical, and any disruption could cause the loss of the entire mission and had to be worked out with great ingenuity by the Mission Control Center (MCC). The best solution against failures was to introduce some degree of redundancy where possible. Sometimes, the complexity of the systems involved allowed for a small amount of flexibility: for example, the Lunar Module (LM) was used as a lifeboat by the astronauts of the *Apollo 13* [1, 2]. However, even though this alternate use of the LM was included in the design and in the mission requirements, *Apollo 13* revealed that the LM carbon dioxide removal canisters were incompatible with the canisters of the Command Module (CM), with possible catastrophic consequences. This example shows how most of the times the design of a mission lacked a holistic view of the entire architecture, which would encompass flexibility involving separate systems and analysis of the impact of systems failure and systems replacement.

Later, as technology evolved, problems in systems engineering grew increasingly complex, and the objectives became more and more demanding, the approach to space systems design changed again. New concepts such as constellations of spacecraft, distributed systems, and modularity were introduced. Nonetheless, the rapid evolution in the aerospace field, the cuts to the space budget throughout the world, and the presence of multiple stakeholders require more thorough consideration of various aspects that did not have enough weight in architecture and design of space systems to date. Traditional systems engineering methods are not always suitable when designing and architecting complex space systems and need to be supported by methodology capable of addressing the new challenges. This is particularly true in the context of Systems-of-Systems (SoS), where the constituent systems have, at least in part, operational and managerial independence [3, 4]. In SoS, dependencies between component systems most affect the behavior of the whole entity. Since the elements are often designed and developed independently, the aggregate emerges only through interaction of components. DeLaurentis and Callaway [5] underline the de-

pendency of functionality on linkages between components in a SoS. Moreover, the behavior of most of the modern space architectures (including mission, logistics, and systems) cannot always be directly evaluated based only on the behavior of the individual systems or subsystems. The whole architecture can exhibit emergence [6–8], meaning that it can show qualities that are irreducible to the constituent parts and arise from the interaction among the component systems. The effect of the dependencies is difficult to evaluate by simple inspection of the systems during design. In addition to the number of systems involved, their complexity, and the importance of the dependencies among them, SoS engineers must also account for the presence of multiple stakeholders and external decisions makers, including international and commercial partners in the case of space exploration. Finally, when complex collections of systems are involved, it is important to evaluate and quantify partial capabilities that can be achieved during the development of the whole SoS.

Due to these features, many modern space missions fall in the category of System-of-Systems. Therefore, some of the specific properties of SoS must be accounted for, in order to achieve better results in terms of operability, cost, reliability, robustness, and resilience. Systems engineers often struggle with complex dependencies among systems and between systems and capabilities to be achieved, in both developmental and operational relationships. A developmental relationship means that the development of a certain system is dependent upon the development of another, but this relationship does not necessarily affect their functioning. For example, the size of the fairing of a rocket depends on the size of the payload, but the operability of the fairing does not depend on the operability of the payload. Instead, an operational relationship means that a certain system needs input (data, material, energy) from another system to operate. In this dissertation, the expression *complex systems* denotes those systems whose collective behavior does not follow trivially from the behavior of individual parts [9] and cannot be easily inferred by separate analysis of these parts. The expression *complex dependencies* denotes those interactions among systems that cause the emergence of collective behavior beyond the simple sum of the behavior

of individual parts. These complex dependencies which, as mentioned above, play a fundamental role in the behavior of SoS are a source of high risk and cost in systems at the leading edge of technology, as space systems are. Cascading effects can cause small failures in individual parts or component systems to have a large impact on the overall behavior. In order to minimize the impact due to the failure of a single launch or to budget reduction, space exploration has been mainly characterized by a series of monolithic missions that are largely independent from one another. In the last few years, the space community has begun to take promising steps to deal with the challenges in space systems engineering. For example, the European Space Agency (ESA) built a facility to apply the Concurrent Engineering method to the design of future space missions [10]. Concurrent Engineering allows for integrated development, parallelization of tasks, and a *top-down* approach, that gives the capability to analyze properties of the SoS as a whole. NASA recently exploited some flexibility in space systems in order to reduce risk. For example, it used the Mars Reconnaissance Orbiter (MRO), whose main mission is to provide mapping of Mars, as a communications relay during the landing of Curiosity rover in the Mars Science Laboratory (MSL) mission. Practitioners in the space community also undertook approaches aimed at increasing the sustainability of space missions, and reliability and scalability of space systems. Spacecraft modularity, on-orbit assembly, network of intermediate exploration missions are among these approaches.

However, these efforts still inherit most of the traditional systems engineering approach, which often comes short in managing SoS-like features. Engineers investigate dependencies between systems by means of computationally expensive simulations, complicated parametric models whose support functions lack intuitive meaning that may ease their use, or analytical tools that add qualitative traits to data that are sometimes already qualitative, thus decreasing the fidelity of the results. Therefore, systems engineers require additional tools and methods to analyze and quantify properties of the system as a whole and to include considerations about possible emergent behavior when designing or updating complex systems [11, 12]. Metrics that describe

the features and the behavior of individual systems do not directly translate to an assessment of the overall behavior. Many authors recognize the importance of holistic high-level metrics and *top-down* approach [13, 14] and acknowledge the need to include thorough analysis of the impact of dependencies between components in the process of designing, architecting, and planning updates of systems and SoS.

It is important to underline that methodologies and tools developed to address SoS features of complex systems, including space systems, and to add a *top-down* perspective to systems design do not have the goal to replace traditional systems engineering methodology. The objective of the necessary new methodologies is to complement and support the *bottom-up* approach, to add holistic considerations about the behavior of large entities as a whole, to improve the analysis and understanding of the impact of dependencies, and to better evaluate features of interest, such as robustness and resilience.

In this dissertation, both terms *robustness* and *resilience* denote holistic features of an architecture, that is a network of interdependent systems, related to a set of operabilities of interest to the user. *Robustness* describes the property of an architecture to maintain a high level of operability of interest following disruption or failures in one or more of the component systems. *Resilience* describes the property of an architecture to react to a loss in operability, for example by re-arranging the component systems, to recover all or part of the lost operability. The formulation of metrics for the robustness and resilience of an architecture in the operational domain (equations 4.1, 4.2, 4.3 and 4.4) is based upon these definitions.

Chapter 2 provides a thorough description of current efforts and limitations found in literature, in support of the preliminary considerations expressed in this introductory chapter, which bring to the conclusion that the systems engineering community needs new tools and techniques to address new issues in architecture and design of space systems and missions. These tools and techniques must allow the users to analyze and quantify the effect of dependencies between systems in a complex space SoS and must support decisions in architecture and design of space systems.

1.2 Goal, objectives and scope of this research

> *"Numquam autem invenietur, si contenti fuerimus inventis. Præterea qui alium sequitur nihil invenit, immo nec quaerit. Quid ergo? Non ibo per priorum vestigia? Ego vero utar via vetere, sed si propiorem planioremque invenero, hanc muniam... Patet omnibus veritas; nondum est occupata."*
>
> *"However, nothing will ever be discovered if we rest contented with discoveries already made. Besides, he who follows another not only discovers nothing but is not even investigating. What then? Shall I not follow in the footsteps of my predecessors? I shall indeed use the old road, but if I find one that makes a shorter cut and is smoother to travel, I shall open the new road... Truth lies open for all; it has not yet been monopolized."*
>
> Lucius Annæus Seneca
> *Epistulæ morales ad Lucilium*[1]*, 62-63 AD*
> *(Book 4, §33,10-11)*

Recognizing the importance of dependencies between systems in a complex SoS, and the existence of SoS-typical features in space systems, this research proposes to develop and test innovative methodologies and tools to perform and exploit analysis of dependencies, both operational and developmental, and to apply these methodologies to examples of space systems.

1.2.1 Research goal

Based on the gaps and needs identified through the extensive literature survey that will be described in chapter 2, the goal of this research is to:

Develop a framework of holistic-level tools and methodologies that address complex systems design process in various fields, including space missions and complex space systems design, with a System-of-Systems perspective; the purpose of these tools and methods is to support more effective decision-making, identification of criticalities,

[1] Moral Epistles to Lucilium

evaluation and comparison of different systems architectures, with the end result of improving robustness, resilience and risk management in space architectures.

This research seeks to achieve this goal through three main objectives:

1. Development of analytical methods to model the holistic impact of dependencies in Systems-of-Systems, both in the operational and in the developmental domain. These methods utilize intuitive parametric surrogate models, that can be used on any general complex network of systems. For this reason they do not refer to field-specific features and therefore are domain-independent. This objective is the core of chapter 3. The methods are:

    - Systems Operational Dependency Analysis (SODA). A parametric model of operational behavior of SoS, SODA can be used for deterministic analysis of SoS, stochastic evaluation of SoS operability, and quantification of the impact of dependencies in case of disruptions.
    - Systems Developmental Dependency Analysis (SDDA). A parametric model of developmental dependencies of SoS, SDDA can be used for deterministic or stochastic analysis of the impact of delays in SoS, inclusion of intelligent automatic scheduling and re-scheduling of development activities, and scheduling decision support.

2. Concepts of use of SODA and SDDA. This objective constitutes part of chapter 3, and the whole chapter 4.

    - Translation of the raw results of SODA and SDDA (operability, criticality, development time) into metrics of interest (robustness, resilience, risk).
    - Employment of SODA and SDDA in small demonstrative applications.

- Integration of SODA and SDDA to include dependency evaluation and SoS considerations in the analysis and synthesis of complex systems

3. Demonstrative application of SODA and SDDA to a complex space SoS. The case study, addressing the problem of Mars exploration, is described in chapter 5.

   - Suggest possible uses of the results of the analysis of impact of dependencies to drive the design, architecture, and development of the Mars exploration SoS.
   - Suggest possible uses of the results of SODA and SDDA to achieve more robust, more flexible, and less expensive architectures, accounting for the SoS-typical properties of Mars exploration mission.

1.2.2 Research scope and contributions

The methods developed in this research can be used to include a top-down approach on a wide variety of applications, including — but not limited to — space systems, defense systems, cybersecurity, smart grids, distribution networks, and transportation. Due to the abstract nature of their mathematical formulation, SODA and SDDA can be easily instantiated for different problems. For this reason, the main goal of this research is the development, description, and demonstration of the SODA and SDDA methods. This includes the reasons for the development of these methods, their roots in previous studies, their mathematical formulation, and advice and suggestions for their application.

Chapter 3 shows how the formulation of SODA and SDDA is unrelated to the features of a specific field. However, the domain independence of these methods requires an additional abstraction effort before the beginning of the modeling and analysis process with SODA and SDDA, as described in chapters 3 and 4. This characteristic suggests that the confidence of the user in the results of analysis with SODA and

SDDA can grow higher if subject matter experts are involved in the modeling effort, especially when experimental or historical data are not available. For the same reason a diverse group, including experts in the various disciplines involved in a specific problem, can have a substantial value when addressing complicated multidisciplinary problems. Furthermore, results of SODA and SDDA analysis to answer questions introduced by experts in one field can suggest additional questions of interest related to a different area of expertise.

To better illustrate the possible uses and the value of SODA and SDDA, I enlarged the scope of this research to some demonstrative applications and to a larger case study in the field of space mission design, which is both my deepest interest and a clear example of the complexity of modern technological endeavors. SODA and SDDA analysis can deal with robustness, resilience, cost, managerial and operational independence, schedule, risk, and stakeholder objectives. However, the scope of this dissertation is limited to robustness, resilience, schedule, and risk. Users of these methods will hopefully envision more and more applications and extensions of the formulation presented in this dissertation.

The present research is meant to bring valuable contribution into the field of architecture and design of complex systems, demonstrated here in the setting of space systems and mission architectures. The following contributions are expected as a result of this research:

- Development of methods and metrics than can assess the effect of dependencies on the development and operability of Systems-of-Systems. In current practice, a thorough analysis of dependencies between the systems involved in a mission is often overlooked, possibly because of a lack of appropriate means to explicitly introduce top-down considerations in systems design. The advantage of considering these issues is twofold: first, the effect of dependencies is taken into account when architecting the System-of-Systems, in order to reduce the criticality and improve the architecture in terms of robustness with respect to degraded operability and to delays in the development. Second, the possi-

ble positive impact of dependencies is used to obtain desired features, such as robustness and flexibility, that might occur only by coincidence if traditional systems engineering methods were used. These methods and techniques help to include these desired features by architecting the assemblage of systems for a mission, i.e. a true System-of-Systems.

- Development of methods capable to quantify partial dependencies, and the effect of partial degradation on the overall capability of complex systems. These methods can improve risk analysis and identification of criticalities in early design phases.

- Development of methods capable to address the complexity, managerial independence, and stakeholders - related issues of Systems-of-Systems. When different stakeholders are involved in the development and operability of a System-of-Systems, they may have different (and competing) objectives. This must be accounted for, when architecting and designing the System-of-Systems, and considerations about multiple objectives are essential.

- Development of tools to guide the future design, architecture, and development of Systems-of-Systems. The analytical methods developed in this research are meant to be used as decision support tools, capable not only to analyze and quantify the features of interest in a System-of-Systems, but also to favor educated decision in development and update of System-of-Systems. Therefore, these tools will have the end result of improving Systems-of-Systems in terms of desired features, such as "quality-to-cost", stakeholders satisfaction, robustness, resilience, flexibility, etc.

2. LITERATURE REVIEW AND PREVIOUS WORK

"If I have seen further it is by standing on the shoulders of Giants."

Isaac Newton
Letter to Robert Hooke, 15 February 1676

Chapter 1 described the background and reasons behind this research, pointing out some of the needs in complex systems engineering. It also underlined how modern space systems exhibit some of the features proper to System-of-Systems, that are often not considered enough when designing and architecting complex space systems.

This section first gives a more formal description of System-of-Systems and their properties. Afterwards, it presents a review of literature on current efforts in system engineering, including the operational and developmental domains, and in space systems design and architecture. This review is obviously not exhaustive, but it spans a good variety of aspects dealt with by this research, and is helpful to identify gaps and limitations in the state-of-the-art approach to systems engineering and needs in the field of complex systems and space systems design. These needs justify and give value to the research goal and objectives listed in chapter 1.

2.1 Systems and System-of-Systems

> *"πάντων γὰρ ὅσα πλείω μέρη ἔχει καὶ μὴ ἔστιν οἷον σωρὸς τὸ πᾶν, ἀλλ᾽ ἔστι τι τὸ ὅλον παρὰ τὰ μόρια, ἔστι τι αἴτιον."*
>
> *"For all things which have a plurality of parts, and which are not simply a total heap, but some sort of whole beyond the parts, there is a cause."* (Often quoted as *"The whole is greater than the sum of its parts."*)
>
> Ἀριστοτέλης
> *Μεταφυσικά (Βιβλίο Η, 1045a, 9-10)*
>
> Aristotle
> *Metaphysics (Book 8, 1045a, 9-10)*

A *system* is a collection of elements that are connected such that they provide functionality that none of them can do alone. A *System-of-Systems (SoS)* can be generally described as a collection of independent and heterogeneous systems that work together to provide capabilities far beyond the reach of any single system. Independence and heterogeneity of the systems in a SoS, as well as other features that I will mention later in this section, are key to distinguishing a SoS from a system. As pointed out in chapter 1, these features pose challenges that make it difficult to address and solve SoS problems using traditional systems engineering approaches. Therefore, systems engineers are recognizing more and more the need to define SoS as a separate class of systems and to increase the effort on developing a structured and comprehensive SoS theory and methodology.

2.1.1 System

While the concept of *system* is intuitively easy to understand, it is surprisingly hard to define. Various authors gave their own definition of system:

- "A system is an assemblage or combination of functionally related elements or parts forming a unitary whole, such as a river system or a transportation system." [15]

- "A system is a set of elements so interconnected as to aid in driving toward a defined goal." [16]
- "A set of different elements connected or related so as to perform a unique function not performable by the elements alone." [17]

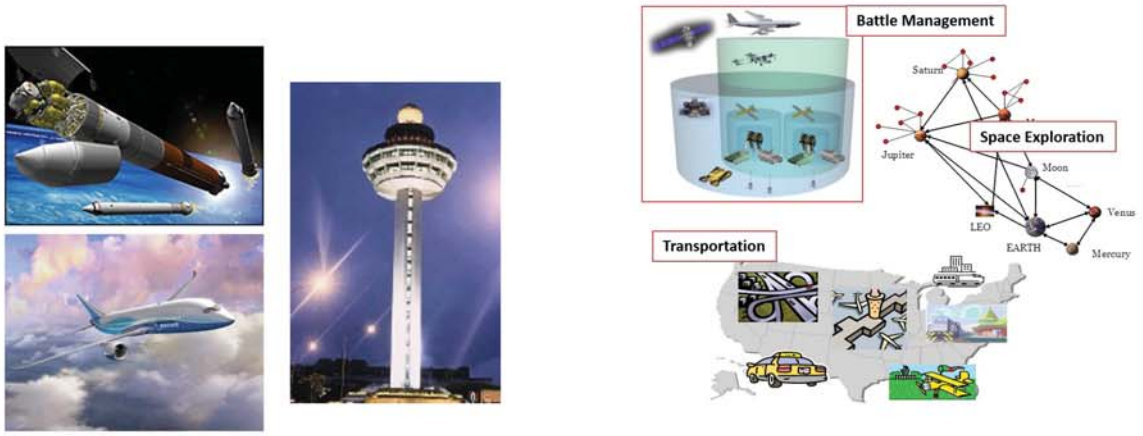
These definitions share three common themes: the presence of a collection of parts or elements, meaning that a systems is composed of multiple subsystems or components. The interdependency between those elements, meaning that elements affect and are affected by other elements. The presence of a certain goal or objective, meaning that every system has an associated utility.

2.1.2 Complex System

Newman [9] defines a complex system as "a system composed of many interacting parts, often called agents, which displays collective behavior that does not follow trivially from the behaviors of the individual parts." He also acknowledges that there is no precise definition for a complex system, but rather a general agreement between practitioners and researcher in the field of systems engineering. This community recognizes traits that distinguish complex systems. Some of these traits, such as emergent behavior and the difficulty of modeling these systems, overlap with some of the definitions of system-of-systems. In fact, systems-of-systems can also be considered a type of complex systems.

2.1.3 System-of-Systems

A System-of-Systems is a particular kind of system, therefore it possesses the attributes of a system, as described above. Yet, due to looser interconnection between the constituent systems and to the amount of complexity involved, it presents more features, and it is even harder to define than a system. Figure 2.1 shows examples of systems and SoS.



(a) Systems (b) Systems-of-Systems

Figure 2.1. Examples of Systems and Systems-of-Systems

The following are some of the definitions that can be found in literature:

- "Systems of systems exist when there is a presence of a majority of the following five characteristics: operational and managerial independence, geographic distribution, emergent behavior, and evolutionary development." [4]
- "Systems of systems are large scale concurrent and distributed systems that are comprised of complex systems." [18]
- "A System of Systems is one comprised of systems whose internal structure and exposed service set is not driven exclusively by the purpose of the larger system, or by each other; otherwise it is just a big, complex system with subsystems." [19]

To better understand the reason for this variety of definitions (and to move beyond), it is useful to briefly describe a history of SoS. Concepts about systems and complex systems as entities that could go beyond the simple sum of their parts arose early in the 20^{th} century, especially during and after World War II, thanks to researchers such as Ludwig von Bertalanffy, Kenneth Boulding, William Ross Ashby, etc.

Bell Labs were the first location where the term Systems Engineering began to be used, during the 1940s. Basic ideas that were the seeds that gave birth to modern day SoS were already circulating in the 1960s. Boulding, for example, considered a collection of theories rather than a theory about individual systems. Despite the early origin, however, these ideas did not catch on and were only sporadically mentioned by other researchers. The first mention of the term SoS for engineered systems came as late as 1989, within the Strategic Defense Initiative [20]. This was followed by an increase in interest in studying SoS problems.

During the 1990s, a new approach to understanding SoS emerged: rather than defining a SoS, which many researcher did within their individual domain of expertise, Maier proposed to use general, distinguishing criteria. Initially, he included five traits that distinguish SoS from a monolithic system [21]. Later, he underlined how two of these traits, the *operational independence* and *managerial independence* of the constituent systems, are the key traits of SoS [3]. Maier's traits are described in table 2.1.

Based on Maier's traits, changing one of more elements of a SoS changes the nature of the SoS. Therefore, the architecture of the SoS, that is the set of constituent components and their interaction, plays a fundamental role in determining the performance of the SoS. When considering systems architecting, engineers must look at both the constituent components and the way they interact with each other. While systems engineering deals with the development of a single product with well-defined requirements, Systems-of-Systems engineering addresses the development of a network of independently operating systems that may collaborate. Such systems usually display emergent behavior (either good or bad), and failures may entail cascading consequences on cost and capability.

Some authors describe the different approaches in system engineering and SoS engineering. Sindiy states that the SoS engineering approach is not meant to generate optimal solutions to specific instances of a problem, but rather to identify positive and negative aspects of architectures and designs, as well as to assess the impact of

Table 2.1. Maier's distinguishing System-of-Systems traits

Trait	**Synopsis**
Operational independence	Constituent elements have their own useful purposes outside the SoS
Managerial independence	Constituent elements operate independently and are provided unique purposes by owners and operators
Geographical distribution	Constituent elements are physically distributed, and linked by exchange of information
Evolutionary development	A SoS is constantly changing, as elements are added or removed
Emergent behavior	Properties of the whole SoS emerge from the assembly and interaction of elements

decisions [22]. This consideration is also a baseline of this dissertation, and is the reason why the methods developed in this research are applicable to generic classes of problems, while specific metrics and techniques are used, in support to the analytical methods, in order to address particular applications.

Keating et al. [11] describe the fundamental differences between systems engineering and SoS engineering, identifying research directions, gaps, and implications for design and development of SoSs. They identify the problems due to the limitation of traditional systems engineering focus, and to the lack of complete, structured analysis at the higher level. They also recognize the need for new strategies and tools to address the complexity, variety, and novel needs of SoS engineering. Furthermore, they point out that these tools must add to classical systems engineering analysis and design techniques, rather than completely replace their use. New methods and

considerations shall still allow the use of classical systems engineering development for systems involved in the SoS.

Due the nature itself of SoS, in order to address their high-level architecture, development, and transformation it is fundamental to consider various dependencies between the constituent systems. These dependencies, that for example can be operational, developmental, physical, of service, of communication, can also have a hierarchy, and an evolution over time. The Department of Defense of the United States of America developed a framework, called Department of Defense Analytical Framework (DODAF) to describe these architectures [23]. The framework is another example of the multi-faceted nature of SoS engineering, and can be used as a base to represent various aspects of a problem, in order to develop methods to address it. Garrett et al. [**?**] suggest the use of DODAF as an overarching comprehensive model that facilitates the ability of the managers and stakeholders to share information. DODAF architectures, often converted into simplified networks, are used to achieve what the authors call "managing the interstitials". Similarly to other authors, Garrett et al. note that designers and decision makers involved in SoS architecting cannot consider only the set of the systems involved in the SoS, but must keep into account their interrelationships. Bonanne [24] suggests a framework based on the Systems Modeling Language (SySML) to model interactions and dependencies within a SoS. Nai Fovino and Masera [7] further underline the importance of the dependencies in SoS. They show how, due to the dependencies, degraded operability in a systems could not be critical for the system itself, yet cause the emergence of a disservice in another system, or at a different level in the hierarchy of systems. These consideration are the foundation of the methods developed in this research.

Other authors characterized SoS engineering perspective at a high level. Rhodes et al. [13] propose a framework to develop and modify a SoS architecture over time, accounting for complexity, policies, stakeholders, and above all for the evolution of strategies and requirements over time. The evolution of the SoS architecture follows a path of changes, with the objective to reach performances as close as possible to those

of a *utopia path*. Dahmann and Smith [12] recognize the same needs, and suggest a *wave model*, describing the evolution of a SoS architecture as a chain of iterations cycling through SoS analysis, development of architectures, and updates based on the needs and requirements of the stakeholders, and on the properties of the current architectures. Even in this case, however, tools and methods need to be developed to execute these steps.

To address some of the gaps found in SoS Engineering, and to perform the analysis required in the SoS approaches, the first step involves determining metrics that can quantify the various and multidisciplinary features of a SoS. Beesemyer et al. [25] tried to frame relationships between decisions in SoS updates and some of these metrics of interest, that they call *ilities*. Their conclusion is that, despite the desire to implement these *ilities* in designs, the number and extension of methods for identifying "good" designs with respect to preferred *ilities* is still inadequate. Furthermore, there is no consensus in the systems engineering community about the tradeoff between such features. This problem is addressed by the Weck et al. [14], who investigate relationships of coexistence and hierarchies among twenty *ilities*. Also Uday and Marais [26], in a multidisciplinary review about resilience in SoS, identified relationships between resilience and other metrics, such as robustness, reliability, pliability, flexibility.

2.1.4 System-of-Systems signature area at Purdue University

In 2003, the College of Engineering at Purdue University announced the creation of eight *signature areas*, i.e. multidisciplinary research initiatives that can cut across the boundaries of engineering and related disciplines. These signature areas were selected among those that could address national priorities and present opportunities for groundbreaking research. One area proposal led by Dr. William Crossley was System-of-Systems. The creation of the SoS signature area afforded many opportunities for others to contribute, for example it produced the establishment of a

System-of-Systems laboratory, led by Dr. Daniel DeLaurentis, in the School of Aeronautics and Astronautics at Purdue University.

The research presented in this dissertation has been conducted in the System-of-Systems group at Purdue University, therefore it also inherits from previous and current research performed by this group. Relevant research from the SoS laboratory at Purdue University [6,22,27–38] will be described in the appropriate sections in the remainder of this chapter.

2.2 Systems operational domain

> *"Consequences are unpitying. Our deeds carry their terrible consequences, quite apart from any fluctuations that went before—consequences that are hardly ever confined to ourselves."*
>
> George Eliot
> *Adam Bede, 1859 (Chapter XVI)*

2.2.1 Operational dependencies and behavioral models

Current practice uses various approaches to model and analyze complex behavior due to dependencies. Response Surface Methodology (RSM) [39] models the relationship between input and output variables with a polynomial. This methodology was widely modified and improved for years [40], and evolved into Robust Parameter Design (RPD) [41]. These techniques result in detailed modeling of systems behavior, but the parameters of this model do not have intuitive meaning, in the sense that they represent a function but are not directly related to features of the dependencies. Therefore, extensive simulation is required for analysis, and the design space exploration is usually executed by generating and analyzing all possible designs.

Bayesian Networks (BN) [42, 43] deal with probabilistic dependencies between nodes in an acyclic network, and are suitable to analyze interdependencies. In the

SoS laboratory at Purdue University, Han et al. [34] developed a model to achieve a more accurate analysis of the effect of the dependencies on the overall behavior of a SoS. Their approach, which uses Bayesian Networks to identify criticalities in networks of functional dependencies, has several advantages. A failure in a system will affect all its dependent nodes. The operability of a system includes both its reliability and the effect of the dependencies. The method is dynamic, and models the evolution over time of the probability of failure in the systems, as well as the resilience of the entire SoS. However, the regression analysis required to identify the correct conditional probabilities is computationally (or experimentally) expensive and may require excessive time in the case of large networks. In addition, these family of methods also gives a very detailed model of the systems behavior, but it does not readily provide information on the root cause of observed behavior, so that improvement of given architectures and design space exploration is not well informed.

To overcome these limitations and to deal with the complexity of large systems, other network-based methods have been proposed, with the goal of facilitating the understanding of complex systems and SoS architectural features. Some of these are specific to a particular class of problems. For example, the Generalized Information Network Analysis (GINA) method models the flow of information in aerospace systems for communication and sensing [44], with the goal of assessing cost, capability, and performance. A common framework used in systems engineering to deal with dependencies is the Design Structure Matrix (DSM) methodology. Analogous to adjacency matrix in graph theory, DSM is used to model and analyze system structural features and dependencies [45, 46]. DSM uses both metrics from graph theory, for example betweenness centrality, and ad-hoc algorithms for clustering systems and support organization in system development. DSM has been extended to multiple domain, with the Multiple Domain Matrix (MDM) methodology [47], and successfully applied by Bartolomei et al. [48] to Engineering Systems. Engineering Systems Multiple Domain Matrix (ES-MDM) constitutes a conceptual framework to model in-

teractions between systems, functions, objectives, activities, for application of DSM analysis methods.

As a parametric model, SODA can be used in conjunction with existing frameworks for system analysis. For example, SODA adds more insights into the impact of the dependencies between systems in the operational and functional domain, and can therefore be used in conjunction with DSM and ES-MDM framework. DSM and ES-MDM support SODA users in deciding the adequate set of nodes required to model the system behavior, and constitute a valid tool to represent matrices of SODA parameters and to identify clusters in SODA networks. SODA can identify criticalities inside and among clusters, and suggest ways to shape the dependencies, and improve operational architectures. Another methodology suitable to be used together with SODA is under development in the SoS laboratory at Purdue University. The Robust Portfolio Optimization (RPO) methodology [37,38] outputs selections of appropriate systems to assemble a SoS with given capabilities, based on associated performance and risk. Therefore, RPO can be used to give SoS architectures in input to SODA, and SODA can support better evaluation of the interactions between systems, required by RPO, in an iterative process.

2.2.2 Criticality and risk analysis

Similar approaches to those presented in the previous section, involving extensive simulations, or qualitative assessment and binary dependencies, are used to identify critical nodes in interdependent networks [49–52], and to support decision in risk management [53, 54]. Besides providing a simple model of the behavior of complex networks in case of failure, SODA addresses other limitations of this current approach to failure and risk analysis. First of all, in current practice, failure modes are often considered only at the end of the design process. Kurtoglu et al. [55] underline the need to consider functional failures in the early design process, though the methodology they propose relies on simulations, which require extensive previous knowledge of

possible architectures. SODA constitutes an alternative approach, to similarly consider failures in early design, and to evaluate the impact of system architecture on operability, while reducing the need for simulations. It allows the designer to include the impact of dependencies and identify root causes of the observed global behavior, when failures occur, at the beginning of the design process, replacing simulations with a parametric model of the behavior. Therefore, with SODA the designer can include considerations about risk when selecting the most promising architectures.

SODA also has one more advantage in dealing with risk analysis: methods like Fault Tree Analysis (FTA) [56,57] and Failure Mode and Effect Analysis (FMEA) [58] have limitations when dealing with multiple complex failures, and use ordinal scale (FMEA) or binary faults (FTA). Wang et al. [59] quantify the reliability importance of components in complex networks, through analytical definition of failure criticality index. This approach accounts for the dependencies in the network, as well as for the reliability of each components, with similar results as FTA. While these systems engineering approaches are still valid when dealing with complex entities like SoS, the capability to evaluate the impact of partial failures and to give a quantitative value to this impact may be more useful as complexity increases. This capability would allow system engineers to address only the criticalities that have the largest negative consequences on the overall behavior, when the resources are not enough to deal with all the criticalities. SODA extends FTA and FMEA since it addresses the analysis of the effect of system failures accounting for partial failures, and is able to model details of the interactions including multiple failures and cascading effects.

2.2.3 Trade space exploration and architecture generation

Formal trade space exploration in complex systems has been addressed by multiple authors. McManus et al. [60] introduced a framework that allows for rapid architecture selection in the conceptual design phase. However, the qualitative considerations on risk and utility are based on classical systems engineering methodology,

and do not account for some complex features, for example operational dependencies. Other authors include dependencies between variables in conceptual design [61], or in change propagation [62], but only as a qualitative flow, or very simple model of binary or qualitative interdependencies [63], or with probabilistic assessments requiring extensive simulations [64]. SODA is a valid alternative to existing parametric models, or to simulation-based models. While keeping the parametric model quite simple, SODA accounts for quantitative partial dependencies, and can be used to improve existing frameworks for architecture selection and change propagation analysis. An advantage of this quantitative model is that it gives insight into the causes of the observed behavior, thus allowing for informed architecture generation and therefore improved trade space exploration.

2.3 Systems developmental domain

> *"If a given task depends on the completion of other assignments in other functional areas, and if it will, in turn, affect the cost or timing of subsequent tasks, project management is probably called for."*
>
> American Management Association
> *Management Review, 1966*

2.3.1 Developmental dependencies and risk analysis

Various authors have proposed methodologies to deal with systems dependencies and with the analysis of risk propagation among systems in the developmental domain. In the SoS laboratory at Purdue University, Mane and DeLaurentis [30, 31], and Mane et al. [6, 32] made use of a Markov network approach, meant to evaluate delay propagation before absorption, to assess the effect of dependencies in SoS development. Whereas the method can be used to rank the systems based on the criticality of their impact on the development of the System-of-Systems, it does not

account for partial dependencies, and the delay can propagate only to one of the systems dependent on the delayed system.

Other authors analyze risk propagation in a network of interdependent systems, but the propagation model is often simple, with a binary propagation of risk or delay [54, 65, 66], or a qualitative rather than quantitative description of the impact of delays [53]. These methods constitute a valid foundation upon which to assess the parameters of the dependencies in the SDDA model, which can then be used for more thorough analysis of development schedule and associated risks.

The Design Structure Matrix methodology, already referenced when discussing the literature on operational dependencies, can be used also in the developmental domain. Similarly to what stated for SODA, DSM can support SDDA users in deciding the adequate set of nodes required to model the system development, and constitute a valid tool to represent the matrices of parameters of the SDDA model and to identify clusters in SDDA networks. Adding to this process, SDDA can identify criticalities inside and among clusters, and suggest ways to shape the dependencies and schedule the development of the required systems, in order to improve developmental architectures, and support decision policies to decrease cost and risk.

2.3.2 Development scheduling

Since 1959, industry heavily relied on PERT/CPM techniques for schedule, and analysis of time and cost of projects development. In 1959, Malcolm et al. published the description of the Project Evaluation Research Technique (PERT) [67]. The concepts of expected time and latest time of development are still used with little changes from the original formulation, that assumes absolute developmental dependencies of events, and certain forms of probability density of the expected time, to treat the stochastic case. In the same year, Kelley and Walker added coordination of activities, revision, and analysis of critical path, in their Critical Path Method (CPM) [68]. Studies and modifications have been added to PERT/CPM [69, 70].

Cinicioglu and Shenoy proposed a way to solve stochastic PERT networks analytically, using Bayesian networks [71], and Azaron and Tavakkoli-Moghaddam suggested a method for time and cost trade-off in dynamic PERT networks [72]. All these methods are based on a network where the edges represent absolute dependency, meaning that the completion of one event requires that the previous event are fully complete, without partial overlapping of dependent tasks.

However, as the complexity of systems grows, the management of their development schedule becomes more difficult. In SoS, since the elements are designed and developed independently, and the aggregate behavior emerges only through interaction of components, this interaction will drive part of the development schedule. In such settings, and when a large number of systems is involved, it may not be possible to decompose them into smaller entities to analyze absolute developmental dependencies. The user may need a high-level viewpoint of development, which includes development decisions by independent stakeholders. Moreover, the individual entities under development may exhibit partial dependencies, where only a fraction of the development of a system is dependent on the development of other systems. Thus, a system j can be developed with a lead time, that is a period of development of j occurring before the full development of a system i, on which j is partially dependent.

A literature survey of project management, decision support, and scheduling shows that authors recognize the need for lead times accounting for possible partial dependencies [73,74], and for intelligent scheduling and re-scheduling [75–77]. However, the common approach is to use static values for lead times [78], and reactive re-scheduling, based on the observed behavior of the systems under development, in terms of delays, rather than in the expected behavior. Boehm [79] surveys estimation methods for development schedule in Software engineering. The common approach in this case is to compress schedule by allocating an adequate number of workers to each activity. However, he also underlines the fundamental importance of project networks when there are known developmental dependencies, and when dealing with constituent systems in a SoS. The suggested approach in this case is the traditional PERT/CPM method-

ology [80], using expert data for estimation, augmented by on-demand scheduling and improved visibility of work-in-progress and system status [81]. These publications, while not accounting for partial dependency, or for possible limited work breakdown structure (due to size and complexity of the involved systems), underline the problems arising from uncertainty and large systems, and describe methodologies to estimate development times. Krishnan et al. propose a framework to model overlapping in a basic scenario, though not accounting for possible delays [82]. The model is based on the required exchange of information to maximize parallel development. This framework does not deal with estimation of future behavior, and consequent dynamic programming, yet it constitutes a solid background to evaluate the features of developmental dependencies between systems, for example the parameters of a SDDA model.

A final consideration about complex systems and SoS is that information about the impact of delays on the overall development and about the "right time" to start the development of a system are results which bear more importance than a simple knowledge of the baseline schedule of the development of the complex system. By "right time" I mean a time that allows for trade-off between the exploitation of the possible lead time for early completion of development, or for delay absorption, and the higher cost and risk associated with an early start, and a possible longer time to develop the system. This capability would have direct impact on acquisition of complex systems, and shed light on the approach of concurrency [83–85].

Building upon these considerations, and based on previous research described in this section, I developed SDDA to address some of the needs in complex systems development scheduling and risk management. SDDA constitutes a simple model, suitable to analyze and assess the impact of delays in the development, identify criticalities, deal with partial dependencies and their features, and dynamically perform intelligent scheduling and re-scheduling, based on the features of the dependencies, and on the amount of acceptable risk, given as input by the user. Compared to existing methods, such as PERT/CPM, SDDA provides more specific insight into the

effects of multiple and diverse dependencies on the development of systems. The advantages include:

- The beginning time of development of a systems hinges not only on its dependency from the development of another system, but also on the parameters that model this dependency.

- The possible lead time is function of the parameters of the dependency, and can be automatically updated, based on the reliability of predecessors in terms of development according to the schedule.

- The completion time of a system is function not only of the beginning time and development time, but also of the parameters of its dependencies.

- If a predecessor shows low reliability in development, SDDA suggests a lead time equal to 0, i.e. the development of successors must wait until the predecessor is fully developed. The parameters of the dependencies can be used to model the amount of risk that the manager is willing to accept, ranging from policies of high lead times (allowing for more delay absorption, but increasing the risk of long development times and waste of resources) to policies of low lead times (similar to PERT, allowing for delay absorption only outside the critical path, but with less risk of waste of resources).

With the simple parametric model of SDDA, the user can quickly compute the impact of delays and stakeholder decisions on the overall development, based on the topology of the development network, on the features of the dependencies, and on implementation of cases based on possible stakeholder decisions. SDDA can be used in conjunction with methods for the optimization of stakeholder decisions in a SoS. Some of these methods are under development in the SoS laboratory at Purdue University [35, 36]. Based on evaluation of impact and propagation of delays, the user can identify critical systems and dependencies, quantify the robustness of the development network in terms of delay absorption, and evaluate different developmental

choices under various levels of reliability and risk acceptance. The user can also perform trade-off between competing desired features, for example time to complete the overall development, risk, capability to absorb delays, and time spent on each system. The analysis can be repeated during the system development, and the schedule be dynamically re-computed via SDDA, allowing system engineering to deal with the dynamic nature of complex systems and SoS.

2.4 Space Systems Engineering

As underlined in the introduction, space missions and their component systems are growing increasingly complex, and expected to grow further. However, while designers recognize the need for innovative methods in architecting missions and designing appropriate space systems, currently these tasks are still performed mostly based on classical systems engineering techniques. This classical approach leaves many unresolved gaps in the field of space systems architecture and design, since many novel features of these systems are not exploited, nor even accounted for. Space systems engineering is adapting much more slowly than other disciplines to the new needs in design and architecture. Lafleur [86, 87] points out that the space industry has recognized the need for novel features, such as flexibility, to be included in the design. Nonetheless, despite such widespread interest, the state of the art in space systems design and architecture remained mostly unchanged in the last few years, and desirable features of complex space systems and missions are accidental, rather than being generated in the design process.

Innovative approaches to deal with the increasing complexity, cost, and requirements are currently under development in other fields. While space systems usually involve a lower number of systems involved, they are usually complex. Furthermore, the huge cost and the criticality of each mission make space systems more susceptible to risk, and more sensitive to budget cuts. The future of space missions requires features that are not explicitly addressed in current designs, such as flexibility, resilience,

maintainability, reliability. In the future multiple stakeholders, including commercial parties, will enter the space systems market, thus adding new variables to the analysis. Techniques and methods suitable to address these issues must include analysis of the complex systems or SoS at different levels, and combine top-down viewpoint and requirements, that describe high-level whole-system perspective, with bottom-up constraints and solutions. Current practice tends to consider the top-down view only at the beginning of the design process, with each system being developed individually, with implicit accounting for integration requirements. Thus, the impact of the relationships between systems is not fully analyzed and exploited. Tatnall et al. [88], focusing on the design process, compare the classical method, that is sequential design, with centralized design, and concurrent design. Sequential design guarantees the communication between different system developers, and the resolution of conflicts only through a process of iteration between a sequence of specialists working in series. In centralized design, a core team is assigned to the overall system, and specialists designing the various systems and subsystems report to the core team. In concurrent engineering, modern information technology favors real-time information exchange, so that the design environment becomes a common space shared by all the designers involved. However, while the concurrent approach is effective in improving the design process of complex multi-disciplinary systems, it is only aimed at supporting effective communication, and helping the integration of individually designed systems. The requirements of the overall system are still broken down in requirements for the single subsystems, without enough consideration for the effect of the interactions between these subsystems on the overall behavior.

The classic sequential approach for space mission design, or a highly centralized approach, are followed in some of the major references for space design [89, 90]. The overall analysis of the mission is used only to define the requirements, that result in baseline mission concepts. At this point, the systems are individually developed and later integrated, but the possibility of increasing the flexibility, the capability, the resilience of the entire mission through the analysis of the impact of interdependencies

is not embedded in the design. The same stovepipe approach is applied to the development schedule and to the analysis of reliability and risk. In complex systems and in System-of-Systems, the operational and development dependencies between subsystems and between systems may be only partial. Concurrent engineering addresses these partial dependencies, allowing for parallel development of partially dependent designs (through real-time communication). However, the analytical tool used to program the development is often a classic PERT/CPM technique, that accounts only for absolute dependencies [90].

The concept of SoS, however, is becoming more and more recognized and understood in space systems engineering. Caffall and Michael [91] underline the importance of the SoS Engineering approach in space systems design. They define architecture as *the collection of logical and physical views, constraints, and decisions that define the external properties of a system and provide a shared understanding of the system design to the development team and the intended user of the system*, and design as the lower level *details of planned implementation which are defined, structured, and constrained by the architecture.* To achieve better design and architecture, they recognize the importance of interfaces, and define various metrics of interest for space SoS, such as availability, reliability, robustness, safety. Jolly and Muirhead [92] describe the increase in complexity and in the importance of the interfaces as the source of new features in space systems, that can now be considered SoS. While the techniques proposed are still part of the classical systems engineering approach, the authors identify current and future space SoS, like the International Space Station, the former Constellation Program, the Mars Sample Return System, and Human Exploration of Mars. Nilchiani [93] proposes to quantify the flexibility of a space system through six factors, including system features, time windows, uncertainties, and response to change. The author recognizes the limitations of his approach when dealing with enormous complexity, multiple stakeholders, results from various disciplines, and changing requirements. However, this approach constitute a good scheme to drive considerations useful for analysis with SODA and SDDA. Lafleur [86] proposes a different approach to

include flexibility in space systems design. He deals with the evolution over time and need to change the architecture to satisfy the requirements, with budget constraints. However, while the approach is closer to the SoS perspective, since it involves external decision makers for the requirements and constraints on the development and evolution of the architecture, the desired feature, which is flexibility, does not originate from the complexity and the dependencies between systems, but rather from the number and capabilities of individual systems.

Many authors dealing with space SoS discuss availability and reliability. However, while they account for the presence of multiple systems and their interaction, the approach does not consider partial malfunctions and partial dependency, and the availability of a system is a binary variable. More than thirty years ago, Bloomquist [94] reported the results of a study about spacecraft anomalies, meant to identify the most critical subsystems. Partial operability was also considered, but the study did not use any method to assess and identify the role of dependencies between subsystems in the observed anomalies. In [95], Greenberg developed maintenance strategies based on the analysis of the overall operability given by the dependencies, and the use of spare components to be turned on in case of failures of the primary components. Failures are stochastic binary variables. A similar approach is described by Castet and Saleh [96, 97], who apply it to multi-layer networks with complex dependencies, and by Walker et al. [98], who quantify *availability* in picosatellites formations. In SoS, the analysis may be too expensive to be performed at the level of single components, therefore high-level methods that can account for partial failures and model partial functional dependencies may be required. Modularity and rejuvenation of spacecraft by on-orbit maintenance is an evolution of the concept of spare components, proposed in the SoS laboratory at Purdue University [33]. In the same laboratory, Sindiy developed an Agent Based Model (ABM) to apply SoS Engineering approach to space exploration [27,28] and to the Lunar Command, Control, Communication, and Information SoS [22,29]. His approach is based on multiple theoretical considerations, and gives clear insight into all the variables involved in such a complex problem. However,

the ABM is very specific to the problem addressed, thus hard to be modified for different problems. Furthermore, an ABM requires multiple simulations, and identifying the reasons of the results of the simulations is not always a trivial task.

2.5 Identified Gaps

> *"Я многое открыл, что было уже открыто ранее меня. Значение таких работ я признаю только для самого себя, так как они давали мне уверенность в моих силах. Сначала я делал открытия дабно известные, потом не так дабно, а затем и совсем новые."*
>
> *"A lot of my discoveries have already been discovered before. I only see personal significance in these works, as they gave me self-confidence. At first, I've been making long-known discoveries, then not so long-known ones, and then completely new ones."*
>
> Константин Эдуардович Циолковский
> *К каким новым выводам я пришёл, 1928 (Архив Российской Академии Наук, Фонд 555, Опись 1, Дело 84)*
> Konstantin Eduardovich Tsiolkovski
> *What new conclusions I came to, 1928 (Archive of the Russian Academy of Sciences, Collection 555, Inventory 1, File 84)*

This literature review identified several gaps in the current practice of space systems design and missions architecture. Table 2.2 lists the current gaps that will be addressed by the proposed research, and the corresponding effort I made to fill the gap. The identification number of these gaps will be referenced in the applications described in chapter 4.

Table 2.2.: Identified gaps in literature and efforts of this research to fill them.

	Identified gap	**Research effort**
1	Need for novel methods to analyze space SoS and to address their complexity: design processes are still monolithic and stovepiped. Classic systems engineering approaches alone are not enough to address the increasingly complex systems, with different stakeholders developing and operating systems independently, and multidisciplinary problems.	Development of methods capable to abstract properties of systems and subsystems at different levels, and to model the impact of changes in the operational and developmental status of systems on the behavior of other systems. Evaluation of the criticality of individual systems to holistic features.
2	Dependencies between systems can cause unexpected/emergent behavior, that is often overlooked. The exploitation of the positive effect of some of the dependencies is often accidental.	Definition of features of interest and development of methods capable to quantify these features of interest, based on the analysis of the impact of the dependencies. The goal is to achieve a better understanding of such impact in order to exploit the positive effect and be aware of the negative effect of dependencies, both in the developmental and in the operational domain.

continued on next page

Table 2.2.: *continued*

	Identified gap	**Research effort**
3	The methods found in literature do not model partial failures and different level of operability in reliability and risk analysis. The failures are usually modeled with binary variables, and their occurrence result in a complete loss of the operability.	Development of a methodology that models partial failures. When dealing with complex SoS, this consideration allows for quantification of the robustness of the whole structure, that can be designed to maintain an adequate level of operability following degradation of some of the constituent systems.
4	Need for methods that model partial developmental dependencies, and can evaluate developmental architectures, and schedule the development based on external requirements, managerial independency, and partial capabilities that can be reached during development. Evaluation of delays is usually based on PERT analysis, where the developmental dependencies are always absolute, and the schedule is updated based solely on expert judgment.	Development of a methodology that models high-level dependencies, where part of a system may be developed while another system, from which the first one is dependent, is not yet fully developed. The resulting analysis can quantify early achievement of partial capabilities, and partial or total delay absorption, and can be used as a guideline for managerial decisions.

3. DEPENDENCY ANALYSIS METHODOLOGY

"Perché vi sia specchio del mondo occorre che il mondo abbia una forma."

"In order for there to be a mirror of the world, it is necessary that the world has a form."

Umberto Eco, *Il Nome della Rosa, 1980 (Secondo Giorno, Prima)*[1]

Based on the outcomes of the literature review, I propose the Dependency Analysis Methodology, which has the goal of addressing some of the limitations of traditional systems engineering approach when dealing with complex systems. This methodology assesses the effect of dependencies among components in a monolithic complex system, or among systems in a SoS, both in the operational and in the developmental domain. The two tools I developed, Systems Operational Dependency Analysis (SODA) and Systems Developmental Dependency Analysis (SDDA) are a parametric model of the behavior of the system. In both methods, I model the system-of-systems as a dependency network, where nodes represent the constituent systems and the capabilities that the SoS has to achieve. The edges represent the operational or developmental dependencies. These dependencies are modeled with a small number of parameters, which quantify the impact of the dependencies on the behavior of the whole SoS. The representation of a system-of-systems as a network prevents the methods from being domain-dependent and allows for their application across various classes of problems. This approach has multiple advantages:

[1] The Name of the Rose, 1980 (Second Day, Prime)

- It results in a convenient model, with parameters that have an intuitive meaning, directly related to the features of the dependency. Therefore, they give a direct insight into the causes of observed, and possibly emergent, behavior.

- It supports informed decision making in design and update of systems and SoS architecture, reducing the amount of interrogative operations, such as simulation, required to obtain the necessary information.

- The parameters of the model can be quantitatively linked to a range of possible input sources, including (but not limited to) experiments, historical data, and subject matter expert evaluation.

The goal of this methodology is to provide a framework to support decision in systems design, development, and architecture. Using Dependency Analysis methodology, designers and decision makers can quickly analyze and explore the behavior of complex systems in the operational and developmental domain, and evaluate different architectures under various working conditions and policies. Based on the raw results of the tools, the user can quantify various metrics of interest (for example robustness, resilience, and delay absorption), compare architectures based on these metrics, perform trade-off between competing desired features, ascertain criticalities, identify the most promising architectures, and discard architectures the lack the requested features. This way, in early design process and at the beginning of development, promising architectures can be kept into consideration, and improved based on the information given by the model parameters and the observed behavior, thus supporting the process of concept selection and development schedule (figure 3.1).

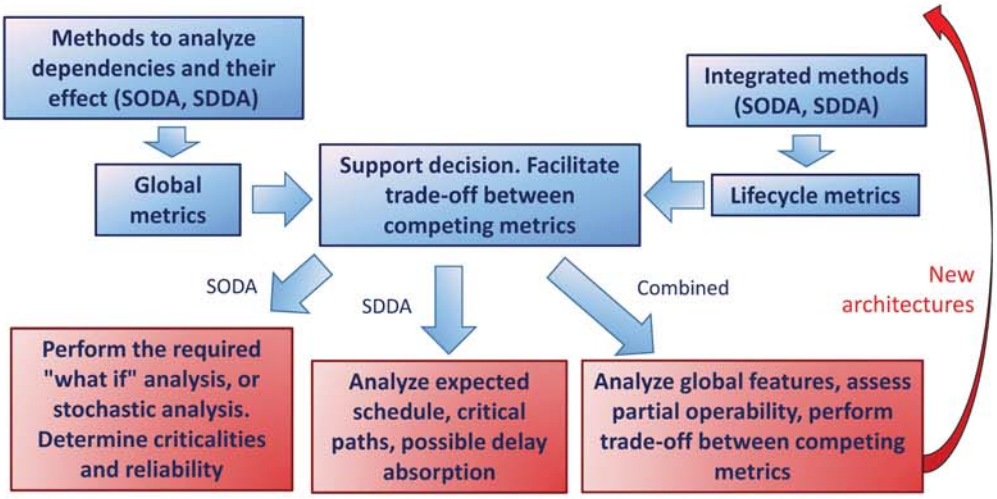


Figure 3.1. Systems Dependency Analysis framework for systems engineering. Separate and combined use of SODA and SDDA.

3.1 Systems Operational Dependency Analysis (SODA)

"La filosofia è scritta in questo grandissimo libro (io dico l'universo) che continuamente ci sta aperto innanzi a gli occhi, ma non si può intendere se prima non s'impara a intender la linrgua, e conoscer i caratteri, ne' quali è scritto. Egli è scritto in lingua matematica."

"Philosophy is written in this grand book — I mean the universe — which stands continuously open to our gaze, but which cannot be understood unless one first learns to comprehend the language in which it is written. It is written in the language of mathematics."

Galileo Galilei, *Il Saggiatore*[2], *1623 (Chapter 6)*

SODA is a method to model and analyze systems input/output behavior in the operational domain. In this section, I first provide a brief comparison of SODA with the existing approaches described in section 2.2. I then give a description of the method and expound SODA mathematical formulation. Afterwards, I list the steps

[2]The Taster

required for problem formulation and SODA application. Lastly, I suggest possible sources of parameters for SODA model.

3.1.1 Comparison with existing approaches

Compared to existing approaches in systems engineering, SODA has advantages and disadvantages. To support the user in evaluating the class of problems to which SODA applies, and the possible uses of this method, tables 3.1, 3.2, and 3.3 summarize strengths and weaknesses of existing methods, and how SODA addresses some of their limitations.

Table 3.1.: Comparison of SODA with other methods for modeling of dependencies

Method	**Features**	**Comparison with SODA**
RSM [39, 40] RPD [41]	Parametric model of systems behavior. Very detailed, but parameters are not directly related to the features of the dependencies. Can require extensive simulation, and the design space exploration is executing by generating and analyzing all possible designs.	SODA is also a parametric model, but parameters have intuitive meaning. Determining the parameters can require less simulation, using information about the systems. In general, SODA model trades high fidelity for the simplicity of the parametric model.

continued on next page

Table 3.1.: *continued*

Method	**Features**	**Comparison with SODA**
Bayesian networks [42, 43]	Stochastic model of dependencies. It is computationally very expensive. Does not provide information about the root causes of observed behavior.	With stochastic inputs, SODA can perform probabilistic analysis of dependencies. The number of simulations required is at most the same as for Bayesian networks, but it can be reduced with the use of Design of Experiments, or based on expert evaluation of the parameters.
GINA [44]	Model of information flow in aerospace systems for communication and sensing.	Similarly to GINA, SODA models the flow of inputs to assess capability and performance. Thanks to abstraction, SODA can be applied to a variety of problems.

continued on next page

Table 3.1.: *continued*

Method	**Features**	**Comparison with SODA**
DSM [45, 46] MDM [47, 48]	Analogous to adjacency matrix in graph theory, DSM and its multiple-domain extension MDM model and analyze system structural features and dependencies. Analysis is performed using metrics from graph theory and ad-hoc algorithms for clustering systems and support organization.	SODA adds insights into the impact of the dependencies among systems in the operational domain. It can be used in conjunction with DSM and MDM, which can support SODA users in deciding the adequate set of nodes required to model the system, and constitute a tool to represent matrices of SODA parameters. SODA can identify criticalities inside and among clusters, and suggest ways to improve operational architectures.

Table 3.2.: Comparison of SODA with other methods for trade space exploration and architecture selection

Method	**Features**	**Comparison with SODA**
Rapid architecture selection [60]	Framework for rapid architecture selection in conceptual design phase. Based on qualitative considerations on risk and utility.	SODA also addresses architecture selection in early design phase, using quantitative measurements of complex features, for example operational dependencies.

continued on next page

Table 3.2.: *continued*

Method	**Features**	**Comparison with SODA**
Method to evaluate dependencies between variables [61–64]:	Various methods to include impact of dependency in conceptual design, and address dependencies in change propagation. Qualitative flow, simple binary interdependencies, or probabilistic assessment requiring heavy simulation.	SODA is alternative to existing parametric models, or to simulation-based models. While keeping a simple parametric model, SODA accounts for quantitative partial dependencies, and allows for informed architecture generation and improved trade space exploration.

Table 3.3.: Comparison of SODA with other methods for criticality and risk analysis

Method	**Features**	**Comparison with SODA**
Methods for identification of critical nodes [49–54]	Approaches to identify critical nodes in dependency networks and to support decision in risk management. They involve extensive simulations, and qualitative or binary dependencies. Risk analysis considered at the end of the design process	SODA provides quantitative analysis of the impact of dependencies, and its simple model of systems behavior addresses complex multiple failures. Criticalities are identified in early design phases.

continued on next page

Table 3.3.: *continued*

Method	**Features**	**Comparison with SODA**
Functional failure reasoning [55]	Simulation-based method to include considerations about functional failures in early design process.	SODA allows the designer to include the impact of dependencies and identify root causes of the observed global behavior, when failures occur, at the beginning of the design process, replacing simulations with a computationally low-cost parametric model of the behavior.
FTA [56, 57] FMEA [58]	Methods to address risk analysis. They are based on binary or ordinal scale, and cannot deal with multiple complex failures.	SODA deals with the analysis of the effect of system failures, including partial failures, and is able to model details of the interactions, including multiple failures and cascading effects.

3.1.2 Description and mathematical formulation

SODA is a method to analyze the result of possible cascading effect of dependencies between systems on the overall operability, in case of disruptions. The method is based on previous work by Garvey et al. [99–101], who proposed Functional Dependency Network Analysis (FDNA), a 2-parameters model of dependencies between capabilities. This method is derived from the Leontief-based Input/Output model for infrastructures [102–104]. Haimes proposed the use of a simple linear model of dependencies, to analyze production and services in infrastructures. Garvey and Pinto suggested a 2-parameters piecewise linear model of dependencies between capabil-

ities. The parameters have intuitive meaning, and the method is used to analyze the impact of failures on the desired capabilities in complex systems. While FDNA results in a good input/output model for capabilities, adding value to a simple linear model, direct application of FDNA to model the behavior of complex systems and the impact of dependencies showed some limitations, which restricted its use to support decision in systems architecture. I appreciate the power of a simple parametric model, and kept the idea of separating the impact of a one-to-one dependency into a critical zone, where very low input is available, and a non-critical zone, where high input is available. However, since FDNA has been developed to deal with capabilities, it does not consider the influence of the internal status of a system. Also, it does not include stochasticity, and it fails to model some example of behavior that can be observed in various applications. For example, FDNA cannot model dependencies where the criticality is rapidly absorbed as a result of small increase in the input. Additionally, when multiple dependencies occur, FDNA can model only substitute inputs (OR-like). To address these limitations, based on FDNA and its previous application, and on results from Agent Based Model (ABM) simulations of problems from different fields of application (for example warfare scenarios, GPS-based autonomous flocking of vehicles, space exploration architectures), I propose the SODA model. SODA is a 3-parameters piecewise linear model, suitable to analyze system dependencies, including partial dependencies, and their effect on system behavior. I recognize that the features of complex systems and the associated computational cost to perform analysis of these systems require analytical methods to keep some inherently qualitative aspect. However, one of the goals of SODA is to keep these qualitative features to a minimum. For this reason, SODA parametric model trades a more detailed behavioral analysis for a quantitative representation of the systems behavior. I already underlined how the parameters of SODA model may be evaluated via parametric regression analysis from experimental or historical data, but also be estimated — if such data are not available — by expert assessment. Tools are under development to support the latter; however, expert evaluation should be kept at a minimum, in

order to reduce the possible introduction of subjectivity into the model. I also recommend to perform sensitivity analysis to evaluate the possible impact of errors in the evaluation of the parameters of SODA model. I validated the model through Agent Based Model (ABM) simulations, executed through the Discrete Agent Framework (DAF) [105, 106], and successfully applied the methodology to aerospace systems.

Operational Dependencies

In SODA, I model the architecture of complex systems and SoS as a directed operational network (figure 3.2) $G = (N, E)$, where N is a set of nodes, and E is a set of edges, where each edge is an ordered pair of nodes. The nodes represent either the component systems or the capability to be acquired. Accordingly, the edges represent the operational dependencies among the systems or among the capabilities. In SODA, each node is characterized by its internal health status, or Self-Effectiveness (SE), ranging between 0 (system not working at all) and 100 (system having maximum performance). Each edge is characterized by three parameters: Strength of Dependency (SOD), Criticality of Dependency (COD), and Impact of Dependency (IOD), that affect the behavior of the whole system-of-systems in different ways. These parameters, described in detail in the next section, can come from expert judgment and evaluation, and they yield a direct insight into the cause of the observed behavior. On the other hand, the parameters of the model can be computed based on historical data, or through a limited number of simulations and experiments.

SODA is used to evaluate the effect of topology, and of possible degraded functioning of one or more systems on the operability of each system in the network. The analysis can be a deterministic evaluation of a single instance of the system status, or a stochastic analysis of the overall behavior. In the deterministic analysis, given the SE of each system, and the properties of each dependency, SODA quantifies the operability O_i of each node, according to the model described later in this section. The operability of a node, ranging between 0 and 100, is defined as the level at which

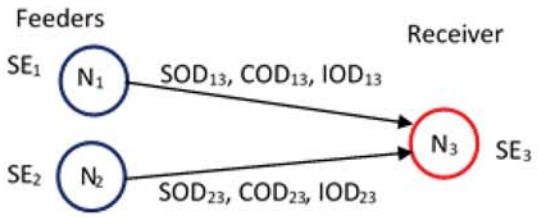


Figure 3.2. Synthetic operational dependency network. N: node. SOD: strength of dependency. COD: criticality of dependency. IOD: impact of dependency. SE: self-effectiveness.

the system is currently operating, or the level at which the desired capability is being currently achieved. The operability of a system is related to performance by means of a given function, as shown in figure 3.3. It is thus related to the value or utility that the system is achieving (however, a value function may include other desired variables than operability). When designing the operational network, the user must also define the relationship between performance and operability. An example of the entire procedure is described in section 3.1.5. In this dissertation, I use a simple linear relationship between desired performance and operability, with $O_i = 0$ corresponding to the worst performance and $O_i = 100$ corresponding to the best performance.

The operability of nodes of interest is used to analyze and evaluate properties of the overall system, such as robustness, resilience, risk. In the stochastic version of SODA, the SE of each system follows a probability distribution. Consequently, also the operability of each node is probabilistic. Differently from the deterministic version, this type of analysis deals with the overall behavior of the system, rather than with a single instance. SODA can thus be used to identify the most critical nodes and dependencies in the network, in terms of impact on the operability, under different disruptive conditions. Designers can compare different architectures, and use metrics based on operability, to quantify robustness and resilience of complex systems, and the risk of each architectural design in terms of impact of disruptions. These metrics may be defined based on the specific application and on the desired

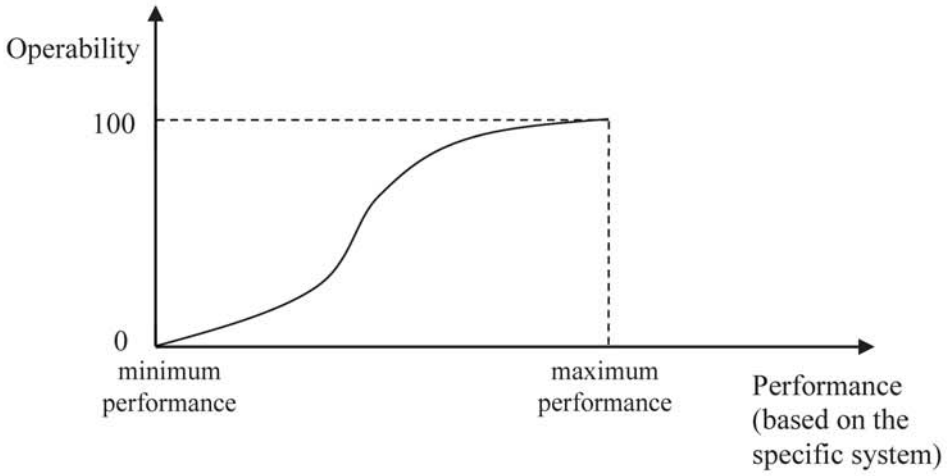


Figure 3.3. Performance and operability. The minimum or worst possible performance corresponds to operability equal to 0. The maximum or best possible performance corresponds to operability equal to 100.

analysis. Equations 4.1 and 4.2 define metrics for *robustness*, based on the comparison between the retained overall operability and the maximum disruption of any system in the network. If flexibility is added to the architecture, by applying rules that allow the architecture to be reshaped in case of disruption (for example, a system can support a disrupted system), I propose to use the comparison between the overall operability when actions are taken to counteract the effect of disruption and the maximum disruption of any system in the network as a measure of the *resilience* of the architecture (i.e. the capability to recover part of the loss in operability). This metric is shown in equations 4.3 and 4.4.

Parameters of the model

Each dependency between two systems is represented with three parameters. Experiments and simulations on real applications showed that this model is more ac-

curate than other models of the same family [100, 103, 104], yet still simple enough to guarantee fast analysis of a high number of architectures. The low number of parameters, and their intuitive meaning, make them both suitable to be assessed by knowledgeable designers, and to be used to drive decision in complex systems architectural design.

Strength of Dependency Strength of dependency accounts for how much the operability of a system depends on the operability of a feeder system. SOD is the predominant factor when the feeder has a high level of operability (SOD zone in figure 3.4). For each operational dependency, the parameter for SOD, α_{ij}, ranges between 0 and 1, and is defined as the fraction of operability of node j that is depending on the operability of node i. The rest of the operability of node j is depending on its SE, that is its internal status. α_{ij} is the slope of the blue line with triangles in figure 3.4. High α_{ij} corresponds to high strength of the dependency.

Criticality of Dependency Criticality of dependency is one of the two parameters that quantify how the functionality of a system degrades when a feeder system is experiencing a major failure. COD is the predominant factor when the feeder has a low level of operability (COD zone in figure 3.4). For each operational dependency, the parameter for COD, β_{ij}, ranges between 0 and 100, and is defined as the maximum loss in operability of node j, that is the drop in the operability level of node j, when node i has operability equal to 0. In figure 3.4, β_{ij} is the difference between 100 and the intercept of the red line with squares with the y-axis. High β_{ij} corresponds to high criticality of the dependency.

Impact of Dependency In the definition of FDNA [99, 100, 107], the slope of the function relating the operability of nodes i and j in the COD zone is always equal to 1. This meant that, starting from the minimum operability of node j (for low levels of operability of node i), in the COD zone this operability O_j would depend solely on the operability of node i, without any contributions by the SE of node j. Results from

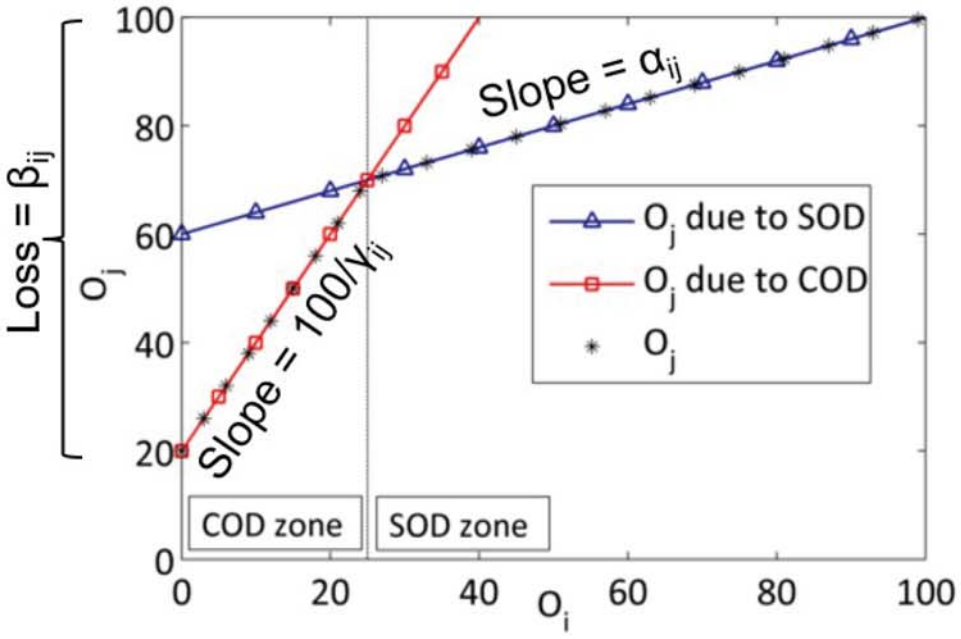


Figure 3.4. Operability due to single dependency of system j from system i. $SE_j = 100$. Blue line with triangles: term due to SOD (here, $\alpha_{ij} = 0.4$). Red line with squares: term due to COD and IOD (here, $\beta_{ij} = 80$, $\gamma_{ij} = 50$). Stars: resulting O_j as a function of O_i. Criticality is prevalent for $O_i < 25$.

ABM simulations showed that, starting from the baseline corresponding to operability of node i equal to 0, the operability of node j can increase faster than the increase in operability of node i. The critical dependency can thus have a lower impact, and be restricted to a smaller zone. A small width of the critical zone models a dependency that may be highly critical, i.e. resulting in a large loss of operability if the input is completely disrupted, but that requires just a small level of operability of the feeder node to achieve high level of operability of the receiver node. Simple linear models and FDNA piecewise linear model fail to capture this feature — which I found to be very common in many systems dependencies — resulting in larger modeling error (figure 3.5). Due to this observation, I introduced a parameter for the Impact of Dependency, γ_{ij}. It ranges between 0 (not included) and 100, and is defined as 100 divided by the slope of the COD-dependent function (red line with squares in figure

3.4). The resulting support function can model a wider spectrum of dependencies than FDNA (for example, it can model dependencies that exhibit an input/output behavior similar to a step function). Other models, for example nonlinear functions, or larger polynomials, may result in a better behavioral model of complex one-to-one dependencies than SODA, but will also increase both the computational cost of the analysis, and the complexity of the model setup.

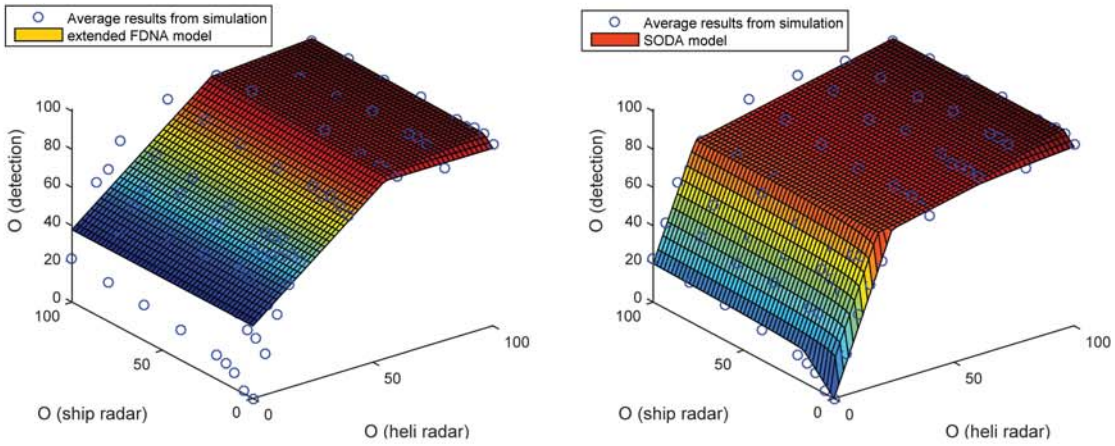


Figure 3.5. Example of fitting the dependency of detection from ship and helicopter radar in a Naval Warfare Scenario. Left: FDNA model modified to consider internal status of systems. The 2-parameters model does not capture the high increase in detection capability caused by slight increase in helicopter radar capability. Also, without the corrective weight described in 3.1.2 ($\lambda = 0$), the zone where the operability of both feeders is low is dominated by the most critical node (helicopter), and FDNA fails to model this behavior. Right: SODA model better captures the input/output relationship, and with the corrective weight ($\lambda = 0.1$) it better models the combined effect of multiple dependencies.

The model: dependency from a single system

Based on the parameters described in the previous sections, SODA models the operability O_j of node j, depending only on node i, according to the following equations.

The operability of root nodes, i.e. nodes that are not dependent by any other node, is equal to their SE, i.e. their internal status:

$$O_i = SE_i \tag{3.1}$$

The operability of a node j, depending only on one feeder node i, is computed as the minimum of two terms (black stars in figure 3.4), one depending on the SOD, one depending on the COD and the IOD:

$$O_j = \min\left(O_j^S, O_j^C\right) \tag{3.2}$$

The term depending on the SOD is computed based on the operability of the feeder node i, and the SE of the receiver node j:

$$O_j^S = \alpha_{ij} O_i + (1 - \alpha_{ij})\, SE_j \tag{3.3}$$

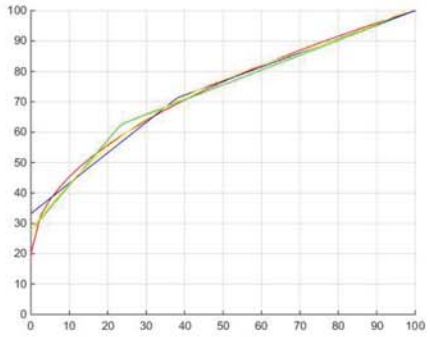
The term in the critical zone is computed based on COD, IOD, and the operability of the feeder node i:

$$O_j^C = (100 - \beta_{ij}) + \frac{100}{\gamma_{ij}} O_i \tag{3.4}$$

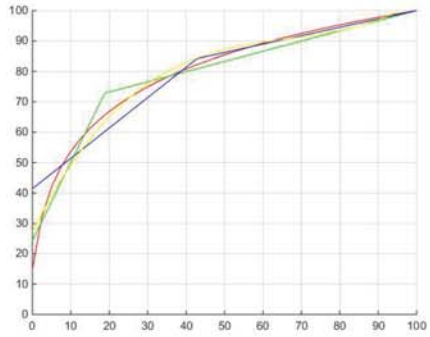
Due to the addition of a third parameter to represent the impact of dependency, SODA results in a better modeling than FDNA. The plots in figure 3.6 compare input/output operability relationship (3.6(a): logarithmic, 3.6(b): square root, 3.6(c): step-like, 3.6(d): user input), represented by the red line, to modeling with FDNA (blue), SODA (green), and polynomials of degree 3 (yellow). The root mean square (RMS) error of SODA modeling is always lower than the corresponding root mean square error of FDNA modeling.

The model: dependency from multiple systems

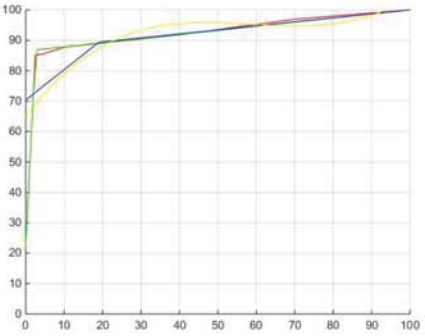
When a node is depending from more than one node, the equations of SODA are slightly modified. Following the suggestion by Garvey and Pinto [99, 100] for FDNA, SODA computes the operability term depending on SOD as the average of



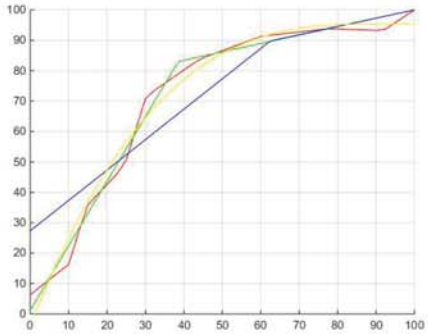
(a) Logarithmic input/output relationship. RMS error in FDNA: 5.08. RMS error in SODA: 2.99



(b) Square root input/output relationship. RMS error in FDNA: 2.45. RMS error in SODA: 1.98



(c) Step-like input/output relationship. RMS error in FDNA: 8.16. RMS error in SODA: 0.51



(d) User input input/output relationship. RMS error in FDNA: 10.08. RMS error in SODA: 2.55

Figure 3.6. Modeling input/output relationships with SODA, FDNA, and polynomials

the corresponding terms for each dependency. SODA computes the operability term depending on COD and IOD as the minimum of the corresponding terms for each dependency, thus reflecting the intuitive idea of the overall impact of a critical dependency. However, the use of this formulation results in a possible non-zero operability of a node even when all the feeders have 0 operability (figure 3.5, left). ABM simulations showed that there are cases of dependency from multiple nodes when operability of a node may decrease to 0 when all the feeders have 0 operability. To keep a simple

model of one-to-one dependencies, I modeled this effect by applying a multiplicative weight W to the parameter that models the COD. For each feeder of the node under consideration, the weight is the average of the operability of all other feeders to the same node, and has an exponent λ ranging between 0 and 1 (in the applications presented in this dissertation, I assumed a value of 0.1 for the exponent, that models a rapid decrease of operability only when all feeders are in critical conditions). With the multiplicative weight, the parameter modeling the COD, γ_{ij}, represents the loss in operability resulting from the total loss of one feeder when all other feeders are working properly. The exponent models how large is the impact of other feeders in the critical zone. An exponent $\lambda = 0$ models an "OR-like" dependency, where each dependency of a node j from a node i results in a certain level of operability, without accounting for dependencies of node j from other nodes. An exponent $\lambda \neq 0$ models a partially "AND-like" dependency, where the operability resulting from the dependency of j on i degrades to 0 if j is not receiving adequate input from its other feeder nodes. Increasing non-zero exponents model increasing sensitivity on the input of other feeders in the critical zone. Multiple dependencies are then modeled as follows.

The operability of root nodes, i.e. nodes that are not dependent by any other node, is equal to their SE, i.e. their internal status:

$$O_i = SE_i \tag{3.5}$$

The operability of node j, depending on multiple feeders, is computed as the minimum of two terms, one depending on the SODs, one depending on the CODs:

$$O_j = \min\left(O_j^S, O_j^C\right) \tag{3.6}$$

The term depending on the SODs is the average of SOD-based terms, computed based on the operability of each of the n feeders, and the SE of node j:

$$O_j^S = \frac{1}{n}\sum_{i=1}^{n} O_{ij}^S \tag{3.7}$$

$$O_{ij}^S = \alpha_{ij} O_i + (1 - \alpha_{ij}) SE_j \tag{3.8}$$

The term in the critical zone is the minimum of terms based on COD, IOD, and the operability of each of the n feeders:

$$O_j^C = \min\left(O_{1j}^C, O_{2j}^C, \cdots, O_{nj}^C\right) \tag{3.9}$$

$$O_{ij}^C = (100 - \beta_{ij})\, W_{ij}^{\lambda} + \frac{100}{\gamma_{ij}} O_i \tag{3.10}$$

SODA for cyclic networks

If the network has cycles, a variant of the same equations can be used. Equations 3.5 - 3.10 constitute a set of non-linear equations, whose solution can be found iteratively, using the SE of each node as initial condition for the operability. Stopping criteria include number of iterations, or change in operability with respect to the previous iteration. However, the applications presented in this dissertation are limited to acyclic networks.

Deterministic Analysis

The simplest analysis that can be performed with SODA method is a single-point, deterministic analysis. Values for the SE of each system, i.e. their possible degraded status, are fed into the equations, to compute the actual operability of each system. This kind of analysis can be thought of as a way to answer "what-if" questions: for example, if the user is interested in the impact of a specific system on the overall behavior of the whole entity, values ranging between 0 and 100 can be assigned to the SE of the system under consideration, and the subsequent operability of each of the other systems can be computed through SODA, and listed in tables or plotted in graphic format. Another example may involve partial failures in several systems, so as to evaluate the combined effect of multiple failures. Deterministic analysis gives good insight into the influence of dependencies into the operational network. The user can identify the most critical nodes under specified conditions, i.e. the nodes that most affect the operability of other nodes. Some of the results may show unexpected

behavior, for example possible reduction of the impact of failures, due to the particular topology. The user can compare different architectures, based on their response to failures and accidents. Once critical nodes have been identified, the user can assess the cause of the observed results, based on the parameters of the dependencies, and thus evaluate possible countermeasures to keep an adequate level of operability. It must be noted that the results of this analysis also depend on the output of interest: for example, a node might be critical on the operability of a low-interest node, while having a small impact over nodes of interest. Stochastic analysis better catches details about the overall impact of disruptions on the operability.

Stochastic Analysis

A more realistic understanding of the systems behavior as a function of the dependencies among components can be achieved by means of a stochastic analysis with SODA. In stochastic analysis, a probability density function of the SE, rather than a single instance, is used. The corresponding output is a probability density function for the operability of each of the component systems, accounting for all the SOD, COD, and IOD values as well as the overall effect of topology. This behavior is condensed in a few probability distributions of interest, instead of long tables, as with the deterministic analysis. In particular, the expected value of the operability of a system gives a measure of the robustness of such system to failures of the feeders, while the variance of the operability evaluates the sensitivity of the system to failures of the feeders. Such outputs show behavioral patterns and features of a whole architecture and thus are valuable in design activities. For example, given the expected distribution of SE of the component systems over time (including aging, minor failures, major accidents), and a threshold for the minimum operability to be achieved by some systems of interest, the user can compute the probability that these systems are operating above the given threshold, as time goes by. The user can then compare alternate architectures, identify their critical systems, and explain the role of topol-

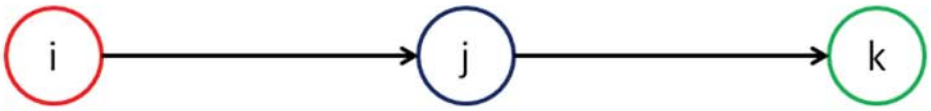
ogy in the observed criticality. Based on SODA equations, an analytic expression for the expected value and variance of the operability can be derived (operability is a piecewise linear combination of the SE). However, due to the possible high number of nodes in the network, and since SODA is computationally inexpensive (table 3.4), Monte Carlo simulation appears to be the best choice to perform this type of analysis. Stochastic analysis, similarly to deterministic analysis, can be used to analyze the combined effect of multiple failures. Since SODA formulation for multiple dependencies does not explicitly treat the correlation between systems feeding the same node, the PDFs of the self-effectiveness of these systems are statistically independent in the model. I suggest two ways to address correlation. The first alternative is for the user to choose input for the Monte Carlo simulation from statistically dependent PDFs. A more accurate alternative is to model the correlation between the systems: since the internal status of the systems is correlated, it means that part of their operability depends on a common cause, which can be explicitly modeled in SODA with a node feeding the correlated systems.

Table 3.4. Computational cost of SODA analysis. Time to perform analysis of a single instance of the operational network, with variable number of nodes and edges. Processor Intel Core i3-2350M 2.3 Ghz, 4Gb DDR3 RAM.

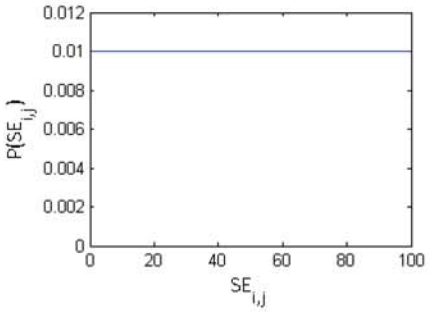
Number of nodes	**Average number of edges (100 runs)**	**Average time**	**Maximum time**
100	2484	0.0125 s	0.0143 s
500	62361	0.887 s	0.922 s
1000	249687	6.625 s	6.804 s
2000	999785	61.41 s	67.82 s

3.1.3 Synthesis and architectural design updates

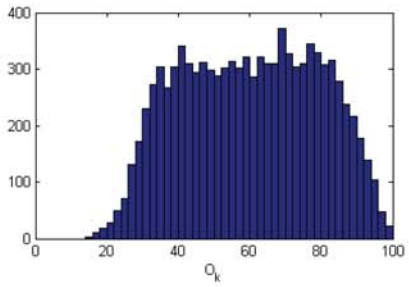
Whereas SODA analysis allows for comparison between various architectural designs, and for trade-off between competing features, the intuitiveness of the parameters used in SODA model can support decision in design updates. Since the parameters give insights into the causes of the observed behavior, they also suggest possible ways to effectively improve this behavior on an already existing architecture, and allow for evaluation of the impact of architectural changes, and modified dependencies. For example, figure 3.7(a), shows a chain of three nodes, i, j, and k. Suppose that the parameters of the dependencies are unknown. The observed behavior, when the probability distribution of the Self-Effectiveness of systems i and j is uniform between 0 and 100 (figure 3.7(b)), and system k is working at maximum Self-Effectiveness, is a distribution with expected value $E(O_k) = 60.1$ and standard deviation $\sigma(O_k) = 19.1$ (a histogram of 10000 instances is shown in figure 3.7(c)).



(a) Simple three-node network



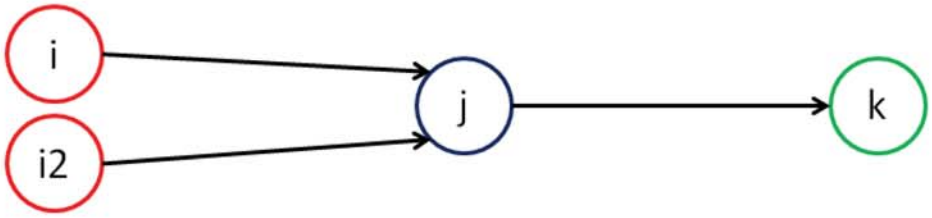
(b) Probability distribution of the self-effectiveness of nodes i and j



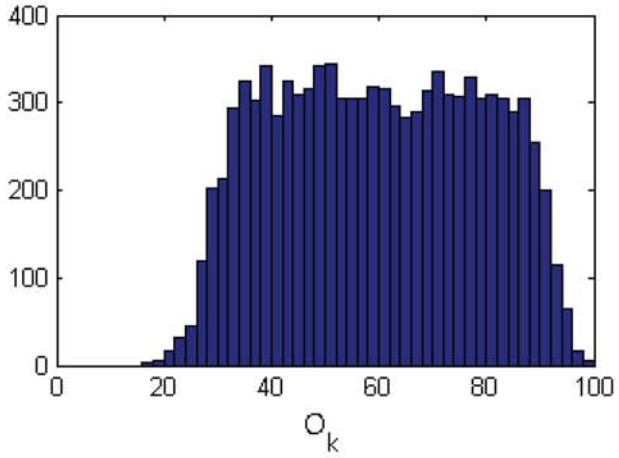
(c) Histogram of 10000 instances of operability of node k

Figure 3.7. Sample use of SODA parameters on a simple three-node network - The network

Since there is no indication about the criticality, the user might test the outcome of adding redundant nodes. Figure 3.8(a) shows the network when node i is supported by a duplicate node *i2*. In this case there is no substantial change: the observed behavior is now a distribution with expected value $E(O_k) = 59.9$ and standard deviation $\sigma(O_k) = 18.8$ (a histogram of 10000 instances is shown in figure 3.8(b)).



(a) The three-node network modified with a redundant node *i2*

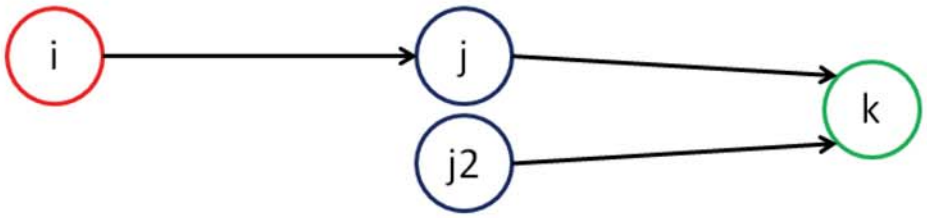


(b) Histogram of 10000 instances of operability of node k

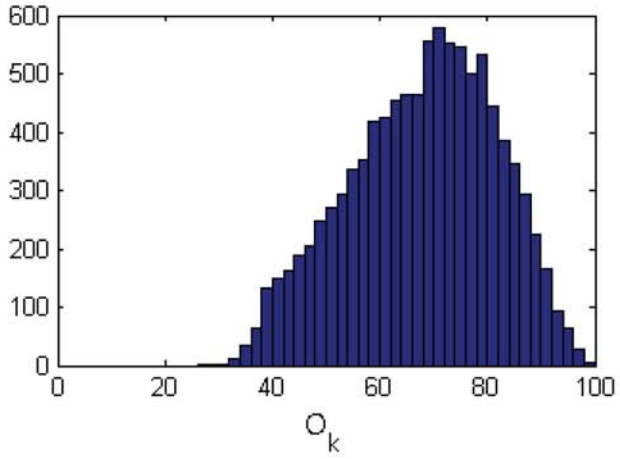
Figure 3.8. Sample use of SODA parameters on a simple three-node network - Redundant node *i2*

Figure 3.9(a) shows the network when node j is supported by a duplicate node *j2*. In this case there is an improvement in the outcome: the observed behavior is now a distribution with expected value $E(O_k) = 67.9$ and standard deviation $\sigma(O_k) = 13.7$ (a histogram of 10000 instances is shown in figure 3.9(b)).

Figure 3.10(a) shows the network when node j is supported by a duplicate node *j2*, dependent on node i. This brings another slight improvement in the outcome:



(a) The three-node network modified with a redundant node *j2*

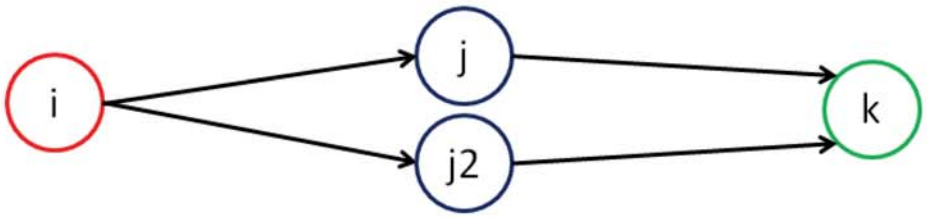


(b) Histogram of 10000 instances of operability of node k

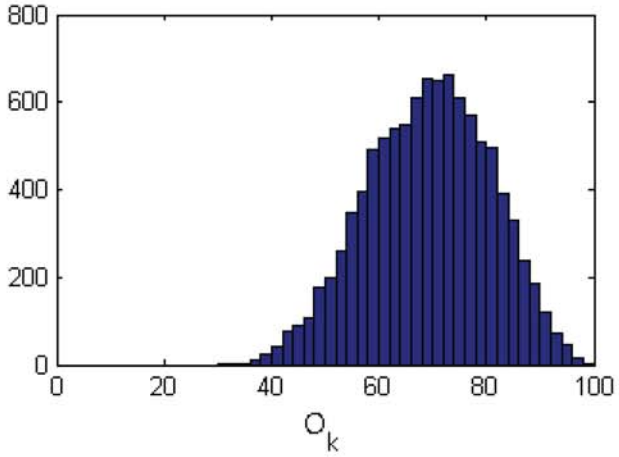
Figure 3.9. Sample use of SODA parameters on a simple three-node network - Redundant node *j2*

the observed behavior is now a distribution with expected value $E(O_k) = 69.2$ and standard deviation $\sigma(O_k) = 11.5$ (a histogram of 10000 instances is shown in figure 3.10(b)).

If the parameters of the dependencies are known, once the criticality of nodes i and j has been identified, the user can evaluate the impact of the parameters of the paths leading in and out of these nodes (figure 3.11). The weakness of node j, its weak dependency from node i, and its strong influence to node k suggest that node j is the main cause of the observed behavior. Therefore, one of the possible improvements that can be implemented is giving some redundancy to this node (this confirms the simulated results). The parameters also suggest that improving the robustness of node i will not have a big impact, since its influence on node j is limited. Other



(a) The three-node network modified with a redundant node *j2* depending on node *i*



(b) Histogram of 10000 instances of operability of node *k*

Figure 3.10. Sample use of SODA parameters on a simple three-node network - Redundant node *j2* dependent on node *i*

improvements may involve increasing the robustness of node *j*, or decreasing the dependency of node *k* from node *j*, if possible.

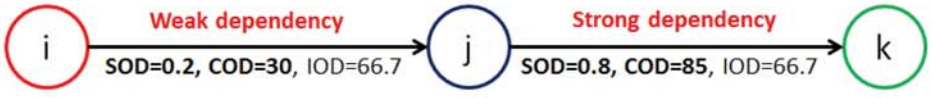


Figure 3.11. Sample use of SODA parameters on a simple three-node network - Parameters of dependencies are known

3.1.4 SODA problem setup

In this section, I list and describe the steps required to apply SODA analysis, and provide guidelines to the use of the method. Since SODA relies on abstraction and generalization, it can be applied to various problems in different fields. However, this capability also entails the need for the user to understand the principles of this methodology, in order to be able to correctly apply SODA analysis to specific problems.

Step 1: operational dependencies The first step necessary to apply SODA is the conversion of the requirements and the systems involved into a network of operational dependencies. It must be underlined that the topology of this network will depend not only on the desired or required topology of interactions among the systems (exchange of information, matter, energy), but also on the desired output from the analysis: the network of operational dependencies will include nodes representing the systems that are assigned to perform a task, and nodes representing capabilities that the user is interested to quantify. SODA is a model of the operational domain, therefore the flow between systems is generalized in terms of operability. Systems Engineering methods for functional allocation can be used to perform this step [108]. Functional Modeling method [109, 110] has the advantage of identifying possible failure modes [111,112], that can also be used as a basis for analysis of the impact of disruptions with SODA. Alternate architectures may be characterized by different systems allocated to perform the required function, different network topology, or different features of the dependencies.

Step 2: self-effectiveness and operability The second step in the problem setup is the definition of the internal status, that will be represented by the SE of each systems, and of the performance, that will be represented by the operability of each node. Both SE and operability are normalized between 0 (worst case) and 100 (best case). Multi-dimensional performance is represented by different capability nodes (and as-

Table 3.5. Example of self-effectiveness and operability for a Naval Warfare Scenario

SoS	**System**	**Self-effectiveness**	**Operability**	**Metrics**
Naval Warfare Scenario	Ship Reconnaissance Systems	Probability of the radar to detect enemy within range	Time to detect the enemy (or % of enemies detected within given time)	Performance, robustness, resilience (flexibility)
	Helicopter Reconnaissance Systems	Probability of the radar to detect enemy within range	Time to detect the enemy (or % of enemies detected within given time)	
	Ship Weapon System	Probability to engage the enemy, when within range	Time to engage the enemy (or % of enemies engaged within given time)	

sociated systems). The measure of internal status and performance, to be associated respectively with SE and operability, can be evaluated by experts, or computed from historical data, experiments, or simulations. Tables 3.5, 3.6, and 3.7 show example of self-effectiveness and operability for specific applications.

Step 3: dependency parameters (α_{ij}, β_{ij}, γ_{ij}) At this point, to be able to model the overall behavior through SODA, the user needs to determine the parameters of each dependency. In their FDNA formulation [99, 100], Garvey and Pinto describe a way to evaluate α_{ij} as the fraction of operability of a node depending on its feeder,

Table 3.6. Example of self-effectiveness and operability for an on-orbit satellite servicing SoS

SoS	**System**	**Self-effectiveness**	**Operability**	**Metrics**
On-orbit satellite servicing	Operational satellite component	Internal operational status	Overall status due to internal status and inputs	Dynamic performance over time, robustness to aging and failures, resilience due to servicing
	Operational satellite	Internal operational status due to component status and dependencies	Overall status due to internal status and inputs	
	Servicing satellite	-	(servicing satellites modify the self-effectiveness of serviced components)	

but accounting for a virtual baseline operability of the system, when the feeders are not giving any input, and there is no criticality. Then they suggest to evaluate β_{ij} as the actual operability that a system achieves, when the feeders are not giving any input. The parameters for the dependencies can be effectively evaluated by experts, based on these definitions and on the equations of SODA, if the users have a good understanding of the impact of each of these parameters on the operability dependency between a feeder node and a receiver node. Figure 3.12 shows the effect

Table 3.7. Example of self-effectiveness and operability for cybersecurity

SoS	System	Self-effectiveness	Operability	Metrics
IT SoS	IT systems	Internal status: generation and transmission of correct information	Generation and transmission / retransmission of correct information, given the internal status, the input, and possible cyberattacks	Performance, robustness, resilience (flexibility)

of α_{ij} on a single dependency of node j from node i, when β_{ij} is equal to 95, γ_{ij} is equal to 10, and α_{ij} increases from the left plot to the right plot. Each plot in the figure represents the operability O_j as a function of the operability O_i. As the SOD grows larger and larger, the operability of node N_j is increasingly dependent on that of node N_i, and less dependent on its SE.

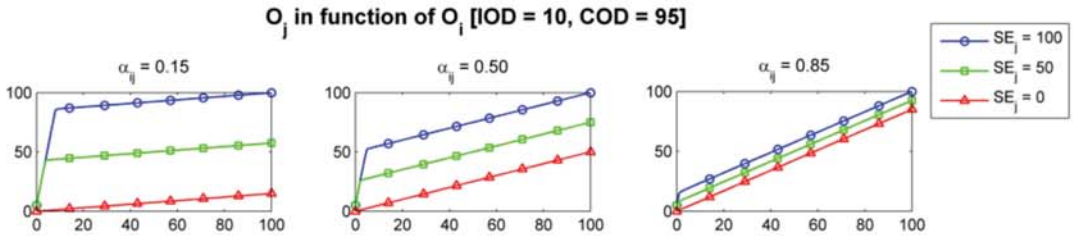


Figure 3.12. Effect of SOD on operability dependency of node j from node i. SOD increases from left to right.

Figure 3.13 shows the effect of β_{ij} on a single dependency of node N_j from node N_i, when α_{ij} is equal to 0.5, γ_{ij} is equal to 40, and β_{ij} increases from the left plot to the right plot. Each plot in the figure represents the operability O_j as a function of the operability O_i. As the COD grows larger and larger, the critical zone (low values of O_i) increases in size, and the loss due to criticality gets bigger.

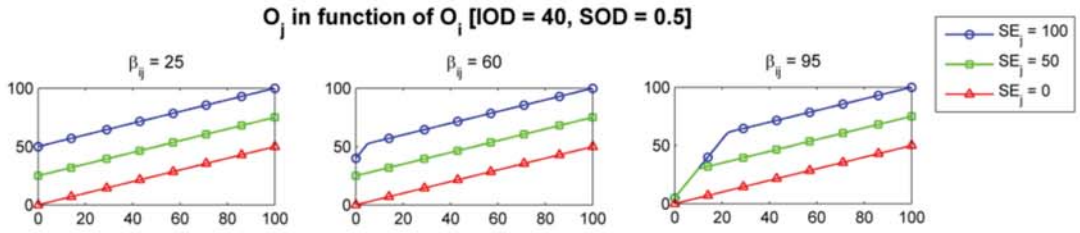


Figure 3.13. Effect of COD on operability dependency of node j from node i. COD increases from left to right.

Figure 3.14 shows the effect of γ_{ij} on a single dependency of node N_j from node N_i, when α_{ij} is equal to 0.5, β_{ij} is equal to 95, and γ_{ij} increases from the left plot to the right plot. Each plot in the figure represents the operability O_j as a function of the operability O_i. As the IOD grows larger and larger, the critical zone (low values of O_i) gets larger, having impact even at high values of O_i, especially when SE_j is high.

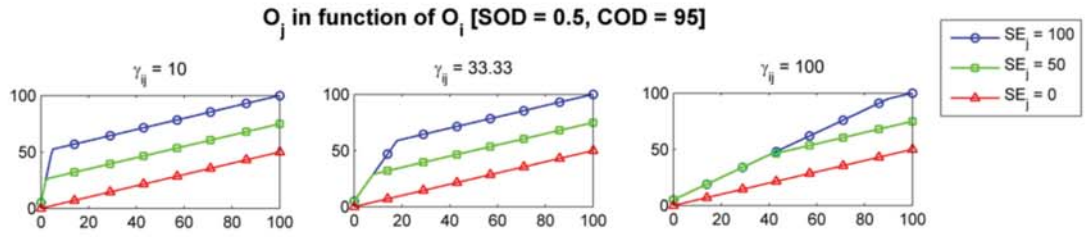


Figure 3.14. Effect of IOD on operability dependency of node j from node i. IOD increases from left to right.

The combination of the three parameters, and of multiple dependencies, results in a model of the systems behavior, according to SODA equations. The process can be

reversed, and historical data, or results from simulation may be used to evaluate, by means of regression analysis, the parameters of the SODA structure that best fits the results. These parameters may then be used to perform more comprehensive analysis, without the need to execute full simulation. Therefore, if the user has historical data or a good knowledge of one-to-one dependency behavior, the parameters that model the global behavior can be computed with a cost of the same order of magnitude as the cost of other parametric models (for example Response Surface Methodology), that is generally lower than the cost of performing full, high-fidelity simulations. If data are not available, Design of Experiments techniques can be applied, so that a low number of simulation is used to identify the parameters of the model. In this case, the cost to compute the parameters is at most the same as simulation-based models (for example Bayesian Networks), while the cost of the analysis is very low (table 3.4). Since SODA is a parametric model, it trades detail for low cost, and for the advantage of having a representation capable to give insight into the reason of observed results. The level of detail of SODA analysis will depend on the level of detail of available data or simulations. For early design phase, I suggest to use simple models, which will be able to identify more and less promising architectures. For later design phases, or for update of existing architectures, more data will be available to support accurate parametric regression.

Step 4: Analysis Once the SODA network, with the relative parameters, is available, the user can perform the required analysis. In step 1, systems and capabilities of interest were included in the network. The user may now decide to perform deterministic analysis, setting up points of interest (each one being a set of SE for each node, corresponding to a specific failure), and use the results to evaluate the impact of different failures, including multiple failures, and to identify the critical systems and dependencies. Otherwise, the user may use models of failure probability in individual systems to perform stochastic analysis and analyze the behavior of the whole entity, and the main features of the impact of failures and functional dependencies. Metrics

of interest, representing for example the robustness and resilience of the entire system can be built based on the results of deterministic or stochastic analysis (systems operability), and their evolution over time, as described in section 3.1.2.

Step 5: Synthesis SODA allows for fast analysis of the effect of various failures scenarios on systems behavior and capabilities, as well as comparison of architectures, and possible trade-off between competing metrics (reliability, resilience, capabilities, cost). In addition, the parameters of the SODA model for operational dependencies give insight not only on the effect of the operational dependencies among the systems, but also on some of the reasons why an observed behavior occurs. Based on the relationship between some of the parameters, the topology of the network, and the observed results, the designer can decide the appropriate actions needed to improve the architecture (increase redundancy, add or delete paths between node pairs, invest money to increase the robustness of critical nodes, etc.). New architectures can be generated based on these observations, and analyzed again by means of SODA. This process can be iterated to improve architectures.

3.1.5 Example of application of SODA steps

Figure 3.15 shows a small Naval Warfare Scenario SoS. The scenario comprises an *MH-60* helicopter, a ship equipped with detection and weapon systems, and an adversary boat approaching the coast. The helicopter and the ship can perform detection of the boat, and the ship can also engage the adversary. In this simple problem, the user is interested in time required to detect the adversary, and time required to engage it. I follow the steps described in the previous section.

Step 1: Operational Dependencies The first step in SODA is the conversion of the requirements and the systems involved into a network of operational dependency. As noted in section 3.1.4, the topology of this network depends not only on the desired or required topology of interactions between the systems (exchange of

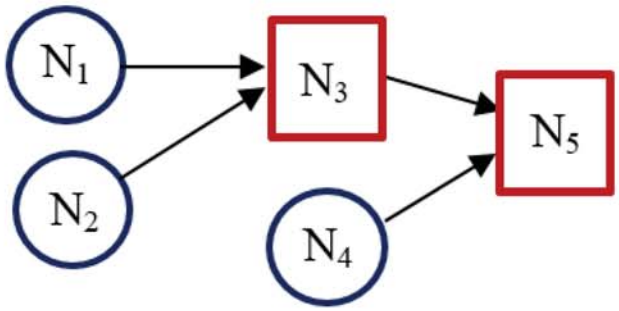


Figure 3.15. The Small Naval Warfare operational dependencies. N_1: ship's radar. N_2: helicopter's radar. N_3: detection of the adversary. N_4: ship's weapon system. N_5: engagement of the adversary.

information, matter, energy), but also on the desired output from the analysis: in this case, two nodes will represent respectively the capability of detection, and the capability of engagement. I use the intermediate node representing the capability of detection because in this application I am interested in modeling the effect of the dependency of detection time from the operability of the detection systems. If the user is interested in different outputs, the nodes of the functional network can be changed accordingly. Three more nodes will represent the helicopter's detection system, the ship's detection system, and the ship's weapon systems. These decisions result in the architecture shown in figure 3.15. Alternate architectures may involve different systems to perform the required function, different network topology, or different features of the dependencies. For example, the user may decide to model the correlation between operability of some systems explicitly. The operability of the ship's radar and weapon systems is likely to exhibit some degree of correlation, since the systems are on the same ship (if the ship is damaged, both systems will be disrupted). For more accurate results, the user could add one node, representing for example the ship hull, feeding both the ship's radar and the ship's weapon system. This node would have a simultaneous impact on the operability of the systems aboard the ship.

Step 2: self-effectiveness and operability The second step in the problem setup is the definition of the internal status, that will be represented by the *SE* of each systems, and of the performance, that will be represented by the operability of each node. In this case, I use simplifying assumptions for detection and weapon systems: they operate within a fixed range, and their *SE* is a measure of the probability to correctly detect or engage the adversary if it is within the operative range. The operability of such nodes correspond to their *SE*, since they are root nodes (they do not have feeder nodes). The operability of the detection capability node is a measure of the time required to perform detection of the adversary boat, normalized between 0 (longest time necessary to detect the adversary. In this case, it is the maximum allowed mission time, before the adversary reaches the coast) and 100 (minimum time required to detect the adversary). The operability of the engagement capability node is a measure of the time required to engage the boat, normalized between 0 and 100, similarly to what I did for the operability of the detection capability node.

Step 3: dependency parameters (α_{ij}, β_{ij}, γ_{ij}) At this point, to be able to model the Naval Warfare SoS behavior through SODA, the user needs to evaluate the parameters of each dependency. Section 3.1.6 describes various sources of the SODA parameters, based on available data or on expert judgment. In this example, since data from a real problem were not available, I ran simulations with an Agent Based Model (ABM), and used some of the results to perform parametric regression and retrieve the required parameters, with the secondary objective of validating the SODA model. If this approach is chosen, proper design of experiment can help to reduce the amount of simulation required to obtain enough data to build the SODA model. I developed an Agent Based Model for the small Naval Warfare SoS, including an adversary boat moving towards the coast (and performing escape maneuvers in case of detection), a helicopter capable of detecting the adversary, hovering over it, and communicating its position to the ship, and a ship capable of detecting and engaging the adversary. I used the Discrete Agent Framework (DAF) ABM to perform a full-

factorial design of experiments, with eleven discrete levels of SE for each system. Due to the uncertainties modeled in the ABM, I ran 10000 instances per each design point. The results, in terms of time required to detect and engage the enemy in various working conditions of radars and weaponry, were used to determine the parameters of the SODA model, with a parametric regression (least square error between the results from the simulation, and results of a model having the SODA structure). A three-dimensional representation of part of the multi-dimensional fitting is shown in fig. 3.5. The following matrices show the parameters identified for the Naval Warfare Scenario SoS:

$$SOD = \begin{bmatrix} 0 & 0 & 0.28 & 0 & 0 \\ 0 & 0 & 0.88 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0.89 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$COD = \begin{bmatrix} 0 & 0 & 9.03 & 0 & 0 \\ 0 & 0 & 85.7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 39.7 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$IOD = \begin{bmatrix} 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 57.6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 52.1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The use of an ABM to validate the SODA model confirmed the need for a third parameter (γ_{ij}) to be added to the original parameters derived from FDNA (α_{ij} and β_{ij}) to better describe some of the behaviors observed, due to the operational dependencies. In FDNA, systems behavior in the region where the feeder nodes are

highly disrupted is not well modeled. In this example, FDNA model resulted in errors of 15% or more in the computed operability. This happens because of the lack of consideration for the effect of other feeders in multiple dependencies, and because FDNA imposes a slope of 1 on the critical portion of the input/output model. To model multiple dependencies in the region of high disruption of the feeders without decreasing the simplicity of the model too much, I found out that including a weight in the operability term depending on criticality (equation 3.10) results in a better model of the systems behavior, capable of modeling OR-like and AND-like multiple dependencies. However, SODA model does not explicitly treats statistical correlation of multiple inputs, that is left to further research beyond the scope of this dissertation.

Step 4: Example of analysis Based on the identified model, I performed analysis of criticality and reliability of the SoS. First of all, I performed a deterministic analysis, imposing a minor disruption (operability degraded to 80%) and a major disruption (operability degraded to 20%) to one system at a time, to identify the systems that have the highest impact on the overall operability. In this case, a measure of the overall behavior is given by the operability of nodes 3 and 5, that represent the capabilities of interest. For this problem, I used the residual operability of the detection and engagement capabilities as measure of the robustness of the architecture, following a disruption. The loss in operability of the required capabilities is used to rank the systems by criticality, with high loss corresponding to high criticality. Results (table 3.8) indicate that this SoS is robust to failures in the ship radar, and even to limited failures in the ship weapon system, while the helicopter radar is the most critical system for detection, with consequent heavy impact on the adversary engagement.

In this kind of analysis, SODA has three advantages over traditional methodologies:

- For very large networks, analysis can be performed in very short time (table 3.4).

Table 3.8. Impact of systems failures on detection and engagement of the adversary

Failed system (minor disruption)	O_3 **(detection)**	O_5 **(engagement)**
Ship radar	97.17	97.17
Helicopter radar	91.25	91.25
Ship weapons	100	91.06
Failed system (major disruption)	O_3 **(detection)**	O_5 **(engagement)**
Ship radar	88.70	88.70
Helicopter radar	49.03	49.03
Ship weapons	100	64.25

- It constitutes a more accurate model than other piecewise linear models (Haimes Input/Output model, and FDNA), capturing effects and behavior that other model would disregard.

- Systems can be ranked based on their impact on the desired behavior, accounting for partial failures and disruption. Traditional approaches, that model only on/off status, would identify the impact of the ship weapon system as the most critical. However, this system is critical only when the failure is total, since it is the only weapon system. Under different circumstances, the helicopter radar is more critical to both detection and engagement capabilities.

I then performed stochastic analysis on the same small network. One example of results is shown in figure 3.16. In this scenario, the SE of radars and weapon system has uniform probability density between 0 and 100. The histogram of a Monte Carlo simulation (20000 runs) is used to plot the probability density of the operability of detection (O_3) and engagement (O_5). Basic metrics related to robustness and sensitivity to failures are shown in table 3.9, assuming a required threshold of $360s$ for

detection, and 600s for engagement. In this case, I use the expected value of operability and the percentage of successful instances following disruption as a measure of robustness of the architecture to the disruption. I use the standard deviation of the operability as a measure of sensitivity to the disruption.

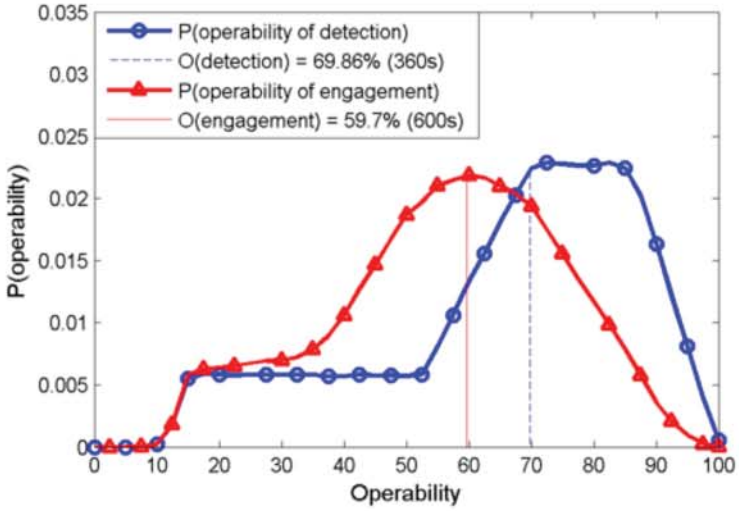


Figure 3.16. Probability density of detection and engagement operability in the small Naval Warfare SoS, when the SE of radars and weapon systems has uniform density between 0 and 100. Blue dotted line: threshold operability corresponding to detection time of 360 seconds. Red dotted line: threshold operability corresponding to engagement time of 600 seconds.

In this kind of analysis, SODA has three advantages over other methodologies:

- For very large networks, SODA can be used to perform computationally inexpensive Monte Carlo simulation, based on known or realistic probability density functions of the SE of each system.
- The use of standard density functions, such as Gaussian or Beta, is not required.
- The global behavior, accounting for the impact of all dependencies, as well as the SE of each system and possible multiple disruptions, can be modeled and analyzed with SODA. Based on these results (and their evolution over time),

Table 3.9. Basic metrics from stochastic analysis: expected value of operability (robustness), standard deviation of operability (sensitivity to failure), percentage of successful instances (robustness)

	Robustness		Sensitivity
	Percentage of instances above threshold	$E(O_i)$	$\sigma(O_i)$
Detection (node 3)	52.8% within 360s	66	21.2
Engagement (node 5)	64.6% within 600s	56.2	18.4

measures of robustness, resilience, and sensitivity can be computed and used to support design of the network evolution.

Step 5: Example of synthesis. Improving the architecture The simple deterministic analysis showed that the most critical system in the network is the helicopter radar. The parameters and the topology indicate that this system is independent from other systems, and the dependency of the detection from this system has very high strength, criticality, and impact. Furthermore, the detection capability has the highest criticality and impact on the capability of engagement of the adversary (meaning that a partial disruption in the detection systems will cause more total delay that a disruption in the weapon system). Stochastic analysis confirms these findings. Figures 3.17, 3.18 and 3.19, and table 3.10 show the resulting probability density for detection and engagement, when the ship detection systems, the helicopter detection system, and the weapon systems have a low-valued internal status, here modeled with a Beta(2,11) density function (appendix A).

Given the analysis performed in this simple case, possible improvement of the architectures may involve:

- Increasing the robustness of the helicopter radar.

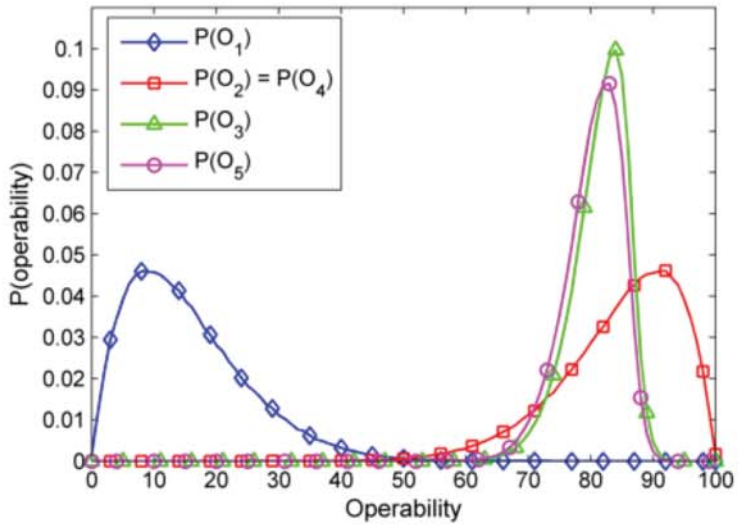


Figure 3.17. Probability density of the operability of each system, when the ship detection system is disrupted. The other systems keep a high level of operability. $SE_1 = Beta(2, 11)$, $SE_2 = Beta(11, 2)$, $SE_4 = Beta(11, 2)$.

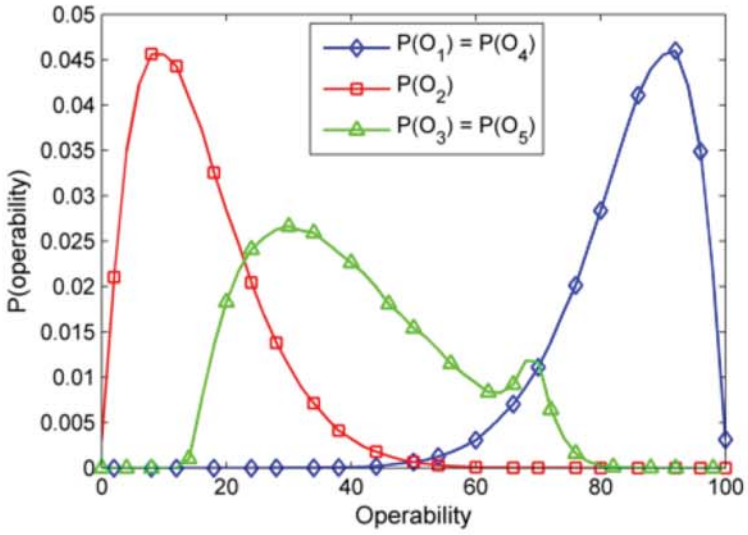


Figure 3.18. Probability density of the operability of each system, when the helicopter detection system is disrupted. The other systems keep a high level of operability. $SE_1 = Beta(11, 2)$, $SE_2 = Beta(2, 11)$, $SE_4 = Beta(11, 2)$.

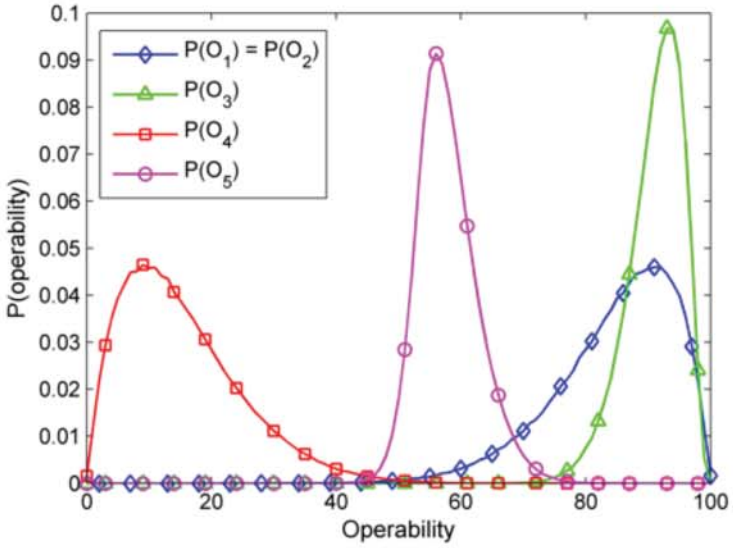


Figure 3.19. Probability density of the operability of each system, when the ship weapon system is disrupted. The other systems keep a high level of operability. $SE_1 = Beta(11, 2)$, $SE_2 = Beta(11, 2)$, $SE_4 = Beta(2, 11)$.

Table 3.10. Expected value and standard deviation of the operability of detection (O_3) and engagement (O_5), when one system is weak (SE modeled with a Beta(2,11) function)

Weak system	$E(O_3)$	$\sigma(O_3)$	$E(O_5)$	$\sigma(O_5)$
Ship radar	81.31	4.43	80.29	4.56
Helicopter radar	40.12	15.1	40.12	15.1
Ship weapons	91.09	4.44	57.73	4.85

- Supporting the helicopter radar with other systems.
- Decreasing the impact of the helicopter radar on the desired capabilities.

Table 3.11 shows the results of an option of redundancy of the radar (that increases the robustness), and an option of a support from an external system aboard the ship, capable to step in for the helicopter radar, in case of degraded capability, at the cost

Table 3.11. Expected value and standard deviation of the operability of detection (O_3) and engagement (O_5), when actions are taken to reduce the impact of helicopter's radar failure

	$E(O_3)$	$\sigma(O_3)$	$E(O_5)$	$\sigma(O_5)$
Original system	40.12	15.1	40.12	15.1
External support	69.87	5.20	69.69	5.17
Redundant Radar	68.63	5.16	68.47	5.12

of a slight loss of SE in the other systems of the ship. I use the difference between the expected operability when actions are taken and the expected operability of the original disrupted system as a measure of the resilience of the architecture (capability to counteract the negative impact of disruption, and partially recover from the loss in operability). A formal metric for resilience is formulated in chapter 4.

The different architectures, that are modeled with different parameters than the original one, show an increased robustness to failures of the helicopter radar. The synthesis of these new architectures must be traded-off with cost.

3.1.6 Source of parameters

This section is meant to provide the user with guidelines about the different possible sources of parameters to model systems behavior with SODA. Since the main focus of this dissertation is the analytical methodology, I used data-based approaches only for some of the small applications described in chapter 4. For most applications, I assessed the parameters based on expert judgment, which is faster but might be less reliable than a data-based approach. When data are not available, and the user must rely on expert opinion to model the input/output behavior of one-to-one dependencies, I recommend to perform sensitivity analysis in order to evaluate the impact of errors in the parameters, and to analyze the behavior of each architecture over a range of the most sensitive parameters.

Data-based approaches

Design of Experiments and data fitting Analogously to the approach followed for other surrogate models, for example Surface Response Methodology, the first possible source of parameters is based on Design of Experiments and data fitting. Once the user has defined the meaning of self-effectiveness and operability for the specific problems, data points can be generated at various levels of operability, and then fitted with SODA model. The choice of the data points can be more or less computational expensive, ranging from Full Factorial Design [113] (blue circles in figure 3.5) to Fractional Factorial Design, for example Central Composite Design [114], Box and Behnken Design [115] and Taguchi's arrays [41]. In general, a higher number of points will result in a more accurate model. The source of the data point can be various: experiments, simulations, and database of historical data are all possible sources of the required data points.

Given a network of n nodes, $G = (N, E)$, each of the k data points is characterized by self-effectiveness level of each node. SE_i^a is the self-effectiveness of node i at data point a. For each data point, experiments, simulation, or archived results give the corresponding data-based operability level of each node, D_i^a. Given a SODA model, characterized by parameters α_{ij}, β_{ij}, and γ_{ij} for each edge $(i,j) \in E$, the same self-effectiveness will result in the corresponding model-based operability level of each node, O_i^a. At this point, the user can perform a parametric fitting, minimizing the square error between the data points and the outcome of the SODA model. The problem is unconstrained but bounded.

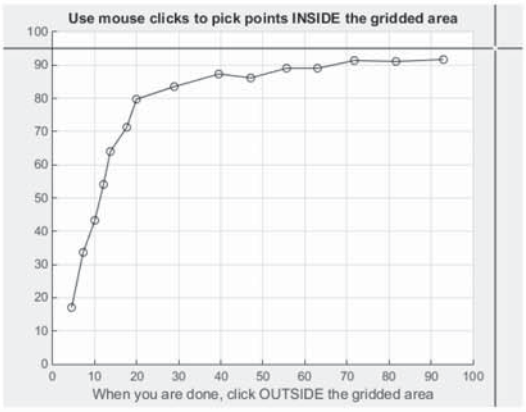
$$
\begin{aligned}
\underset{\alpha_{ij},\beta_{ij},\gamma_{ij}}{\text{minimize}} \quad & \sum_{a=1}^{k} \left\| \begin{bmatrix} O_1^a(\alpha,\beta,\gamma) & \cdots & O_n^a(\alpha,\beta,\gamma) \end{bmatrix} - \begin{bmatrix} D_1^a & \cdots & D_n^a \end{bmatrix} \right\|^2 \\
\text{subject to} \quad & 0 \le \alpha_{ij} \le 1, \ (i,j) \in E, \\
& 0 \le \beta_{ij} \le 100, \ (i,j) \in E, \\
& 0 < \gamma_{ij} \le 100, \ (i,j) \in E.
\end{aligned}
\tag{3.11}
$$

Simple input/output models The data fitting based on Design of Experiments is based on data points generated on the whole system architecture, including multiple dependencies and chains of dependencies. An alternative data-based approach to assess SODA parameters is to use data resulting from simple input/output models (this is effective especially at a lower level of abstraction, down to subsystems and components level), relative to single dependencies between two nodes i and j. In this approach, the data fitting to a SODA model will be performed on one-to-one dependencies rather than on the whole network. Each set of data will result in a SODA dependency model of the type showed in figure 3.4. The parametric fitting is again an optimization problem, minimizing the square error between the data points and the outcome of a one-to-one SODA dependency.

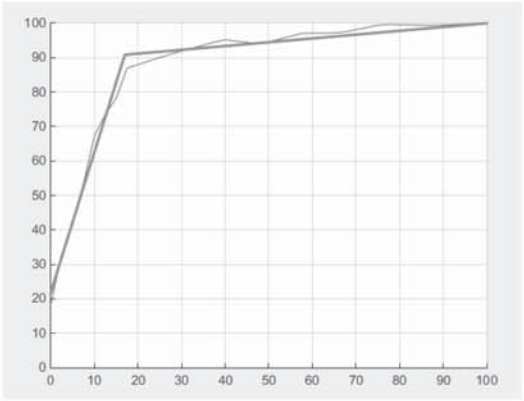
$$\begin{aligned} \underset{\alpha_{ij},\beta_{ij},\gamma_{ij}}{\text{minimize}} \quad & \sum_{a=1}^{k} \left\| \begin{bmatrix} O_i^a(\alpha,\beta,\gamma) & O_j^a(\alpha,\beta,\gamma) \end{bmatrix} - \begin{bmatrix} D_i^a & D_j^a \end{bmatrix} \right\|^2 \\ \text{subject to} \quad & 0 \leq \alpha_{ij} \leq 1, \\ & 0 \leq \beta_{ij} \leq 100, \\ & 0 < \gamma_{ij} \leq 100. \end{aligned} \tag{3.12}$$

Expert judgment-based approaches

One-to-one dependency fitting A similar approach to the simple input/output model is based on expert judgment. Since the model fitting is related to just a one-to-one dependency, an expert user (familiar with the specific input/output behavior on the dependency between node i and j) can place the points that will be used in lieu of data from experiments or simulations in the parametric regression procedure. The regression will follow equation 3.12. Figure 3.20(a) show a *Matlab* interactive figure, which allows the user to input data points for a one-to-one dependency. A *Matlab* script normalizes the values, so the the highest input corresponds to $O_j = 100$, and performs regression to a SODA model. The resulting piecewise linear model is shown in figure 3.20(b).



(a) The points corresponding to behavioral dependency of node j on node i, input by the user on an interactive figure



(b) Fitting of SODA model to the input points, after normalization of the maximum value to 100

Figure 3.20. SODA fitting of data input by expert user for a one-to-one dependency

Parameter selection Alternatively, based on the intuitive meaning of the three parameters of SODA model for a one-to-one dependency, an expert user can directly input appropriate parameters into the model. Figure 3.12, 3.13, and 3.14 can facilitate this decision, by showing the impact of changes in the parameters of the model. In this case, I strongly recommend to perform an evaluation of the sensitivity of the results of SODA analysis to changes in the parameters.

3.2 Systems Developmental Dependency Analysis (SDDA)

"In omnibus autem negotiis priusquam adgrediare, adhibenda est præparatio diligens."

"Before entering any occupation, diligent preparation is to be undertaken."

Marcus Tullius Cicero
De Officiis[3]*, 44 BC*
(Book 1, §73)

SDDA is a method to assess the impact of partial developmental dependencies between components in a monolithic complex system, or between systems in a System-of-Systems. I propose a parametric model of the developmental dependencies. This approach results in a simple, intuitive model, whose output is a schedule of the development of the system, or System-of-Systems, accounting for partial dependencies. Using SDDA, the expected *lead times*, and the beginning and completion time of the development of each system can be automatically scheduled and re-scheduled. SDDA analysis evaluates these times based on the current and expected performance of each system (in terms of development time), on the model of the dependencies, and on the amount of accepted risk. Rules that dynamically change the parameters of the dependencies, based on stakeholder's decision, or on development deadlines, can be added to the basic model for further analysis. SDDA method supports educated decision making in the development and update process of systems architecture. In particular, throughout the whole development phase, the information given by the method can be used to identify criticalities, and bottlenecks, to quantify possible partial delay absorption, and to assess the best time to begin the development of each system, accounting for development cost, decision by other stakeholders, and risk. In this section, I first provide a brief comparison of SODA with the existing approaches described in section 2.3. I then give a description of the method and expound SDDA

[3]About Duties

mathematical formulation. Afterwards, I show how to use SDDA to quantify delay absorption and to support scheduling decisions. Lastly, I suggest possible sources of parameters for SDDA model.

3.2.1 Comparison with existing approaches

Compared to existing approaches in systems engineering, SDDA has advantages and disadvantages. To support the user in evaluating the class of problems to which SDDA applies, and the possible uses of this method, tables 3.12, and 3.13 summarize strengths and weaknesses of existing methods, and how SDDA addresses some of their limitations.

Table 3.12.: Comparison of SDDA with other methods for developmental dependencies and risk analysis

Method	**Features**	**Comparison with SDDA**
Markov network approach to SoS dependencies [6, 30–32]	Evaluates delay propagation in a network of systems. Can be used to rank the systems based on the criticality of their impact on the development of a SoS. Does not account for partial dependencies, and the delay can propagate only to one of the systems dependent on the delayed system.	SDDA can also be used to ranked the systems based on the criticality of their impact on the development of a SoS. SDDA accounts for partial dependencies, and for propagation of delays to all the systems depending on the system which is experiencing the delay.

continued on next page

Table 3.12.: *continued*

Method	**Features**	**Comparison with SDDA**
Delay propagation in network of interdependent systems [53, 54, 65, 66]	Analysis of risk and delay propagation between systems. It is binary (the risk is propagated or not) or qualitative (delay is expected, but not quantified).	SDDA adds quantitative analysis to the propagation of delays. Some of the methods in this category can be used to assess SDDA parameters.
DSM [45, 46] MDM [47, 48]	Analogous to adjacency matrix in graph theory, DSM and its multiple-domain extension MDM model and analyze system structural features and dependencies. Analysis is performed using metrics from graph theory and ad-hoc algorithms for clustering systems and support organization. Some of the matrices tackle developmental dependencies.	SDDA adds insights into the impact of the dependencies among systems in the developmental domain. It can be used in conjunction with DSM and MDM, which can support SDDA users in deciding the adequate set of nodes required to model the system, and constitute a tool to represent matrices of SDDA parameters. SDDA can identify developmental criticalities inside and among clusters, and suggest ways to improve developmental schedule based on analysis of delays and risk.

Table 3.13.: Comparison of SDDA with other methods for scheduling and impact of delays

Method	**Features**	**Comparison with SDDA**
PERT/CPM [67, 68] extension of PERT [69–72]	Methods to evaluate development schedule, based on dependencies between tasks. A task can start only when all its feeder tasks have been completed. There exist at least one critical path from the first to the last task. Delays occurring along the critical path cannot be recovered. Extensions include stochastic PERT, time and cost trade-off, dynamic PERT networks, etc.	SDDA extends PERT by including partial dependencies. Delays can be recovered even on the critical path. Stochastic analysis is possible with SDDA.
Project management, decision support, and scheduling [73–78]	Methods to account for partial dependencies and for intelligent scheduling and re-scheduling. Static values are used for *lead times*.	SDDA performs scheduling and re-scheduling based on the current and expected behavior of the systems under development.

continued on next page

Table 3.13.: *continued*

Method	**Features**	**Comparison with SDDA**
Development schedule in Systems Engineering [79–81]	Approaches for resources allocation, accounting for developmental dependencies. The suggested approach is traditional PERT/CPM methodology, augmented by on-demand scheduling and improved visibility of current system status.	SDDA adds analysis of partial dependencies, evaluation of criticalities, and intelligent scheduling and re-scheduling.
Modeling overlapping of development [82]	Framework to model developmental overlapping, based on the required exchange of information between developmental tasks.	This framework constitutes a solid background to evaluate the features and parameters of developmental dependencies in SDDA model. SDDA adds analysis of impact of delays, and considerations based on the estimation of future behavior.

3.2.2 Description and mathematical formulation

SDDA is a method to analyze the impact of partial developmental dependencies between systems on the schedule and development of the whole complex system or System-of-Systems (SoS). Similarly to SODA, the method borrows the concepts of Strength of Dependency (SOD) and Criticality of Dependency (COD) from previous work by Garvey and Pinto [99, 100], who proposed Functional Dependency Network Analysis (FDNA), a two-parameters model of dependencies between capabilities. Likewise, in SDDA I model the developmental dependencies between systems

with two parameters, having an intuitive meaning that facilitate the modeling process. The parameters are described in detail later in this section. The outcome of SDDA analysis is the beginning time and the completion time of the development of each system, as well as an assessment of the combined effect of multiple dependencies and possible delays in the development of predecessors. The *lead time*, i.e. the amount of time by which a system can begin to be developed before a predecessor is fully developed, is calculated based on the parameters of the dependencies, and the performance of the predecessors. SDDA allows for deterministic or stochastic analysis. In deterministic analysis, SDDA evaluates the impact of a single instance (i.e., one given amount of delay in each system), resulting in one set of beginning time and completion time of each system. In stochastic analysis, the amount of delay in each system follows a given probability density function. Consequently, even the beginning and completion time of each system will be a probability density function. SDDA methodology evaluates the most critical nodes and dependencies with respect to development time and propagation of delays. Results from the analysis are used to compare different architectures in terms of development time, capability to absorb delays, and flexibility.

Developmental Dependencies

The method is most useful for large systems and SoS development networks. These are directed networks $G = (N, E)$, where N is a set of nodes, and E is a set of edges, where each edge is an ordered pair of nodes. The nodes represent systems to be developed. The links, like in PERT networks, represent development dependencies between systems (figure 3.21). A dependency of a system j from a system i means that system j needs some input (information, or other deliverables) related to the development of system i, to be able to complete its own development. Differently from PERT, however, the dependencies are not absolute and account for partial independency of development of each system. Usually, this partial independency is accounted

for by assigning a fixed *lead time*, based on expert judgment. SDDA gives a simple, yet more realistic model of this kind of dependencies.

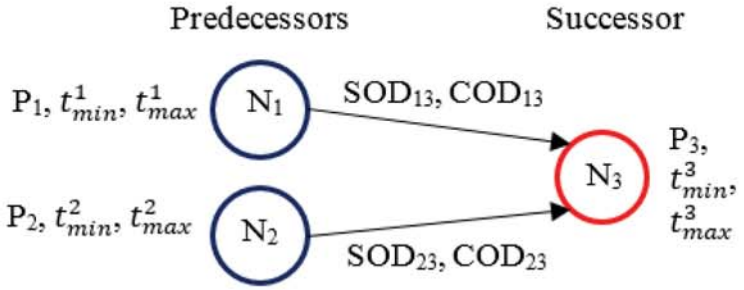


Figure 3.21. Synthetic developmental dependency network. N: node. SOD: strength of dependency. COD: criticality of dependency. P: punctuality. t: development time.

In SDDA, each system i requires three pieces of input data. The first two inputs are the minimum independent development time t^i_{min}, and the maximum independent development time t^i_{max}. These are respectively the minimum and maximum development time of system i, not accounting for the dependencies. The third input is a reliability state, representing the system timeliness, or punctuality P_i. The level of punctuality, normalized between 0 and 100, constitutes an assessment of how much the system is close to be developed in its expected minimum time, not accounting for the dependencies. It is the analogous of SE in SODA. The higher the level of punctuality, the more a system is following the expected schedule, thus resulting in a shorter development time. The relationship between time and punctuality might not be linear. For example, 5 weeks could be the shortest time to develop a system, corresponding to punctuality of 100, 12 weeks could be the longest time to develop the same system, corresponding to punctuality of 0, but 10 weeks could correspond to a *satisfaction* of 50%, therefore a level of punctuality equal to 50. In this research, I used a simple linear correspondence between development time and punctuality (with high punctuality corresponding to short time, and vice versa, as shown in figure 3.22). In deterministic analysis, the punctuality of each system will be a single

number, while in stochastic analysis the punctuality will follow a probability density function. Each link, that is each dependency, requires two parameters: Strength of Dependency (SOD), and Criticality of Dependency (COD), that affect the development schedule of the whole system in different ways. These parameters, described in detail in the next section, can come from expert judgment and evaluation or can be computed based on historical data. The framework to overlap product development activities, proposed by Krishnan, Eppinger, and Whitney [82], suggests a model based on the required exchange of information for parallel development. This approach gives important insights into developmental dependencies, and can be used as a base for SDDA parameters evaluation.

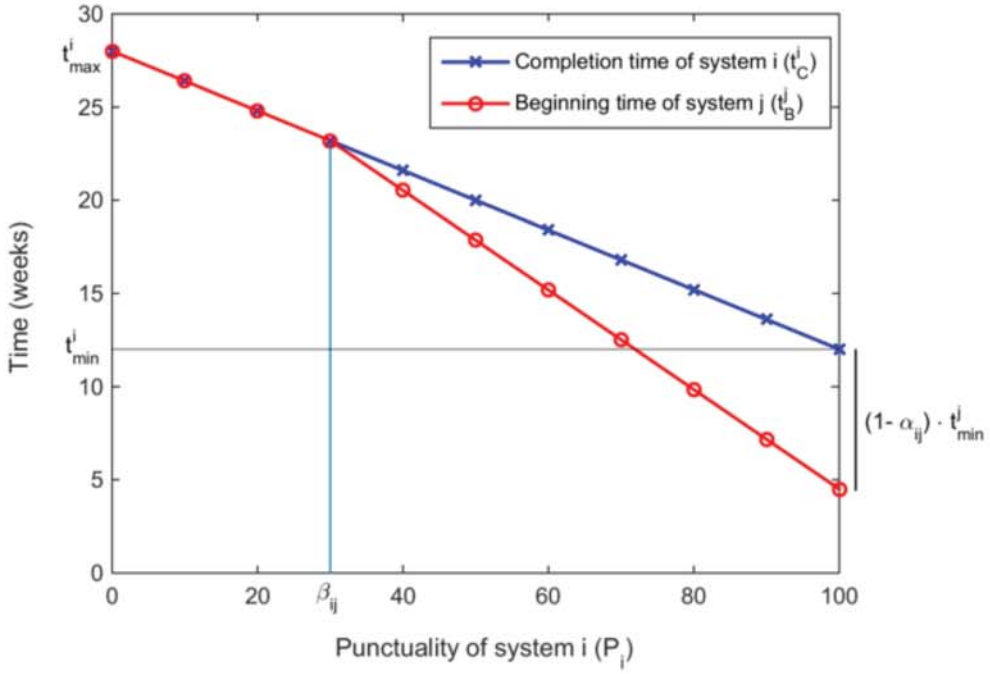


Figure 3.22. Completion time of system i, and beginning time of system j in function of the parameters of the developmental dependency between the two systems ($\alpha_{ij} = 0.25$, $\beta_{ij} = 30$). Due to partial dependency (SOD lower than 1), system j can begin its development before completion of system i, unless the latter is critically late.

Parameters of the model

Each developmental dependency between two systems is modeled with two parameters. The low number of parameters, and their intuitive meaning, make them suitable to be assessed by knowledgeable designers and managers, or based upon considerations on information exchange, as pointed out in [82] At the same time, they overcome PERT/CPM's inability to manage partial dependencies and dynamic *lead time*. Figure 3.22 shows the relation between completion time of a predecessor system i (function of its punctuality P_i) and beginning time of a successor system j.

Strength of Dependency The Strength of Dependency (SOD) evaluates how much the development of a system is dependent upon input from its predecessor. For each developmental dependency, the parameter for SOD, α_{ij}, ranges between 0 and 1 and is defined as the fraction of development time of system i that is depending on inputs by its predecessor j. As shown in Figure 3.22, a system can begin its development earlier than the completion of the development of its predecessor. When the predecessor is developed in its shortest development time (P_i equal to 100), the amount of *lead time* of the successor is equal to its own minimum development time, multiplied by a factor of $1-\alpha_{ij}$. This means that, while the predecessor completes its development, the successor will be able to complete the fraction of its development that does not depend on its predecessor. The value of α_{ij} will trade-off between the risk associated with the decision to begin the development early and the possibility to partially absorb delays thanks to this *lead time*. A delayed development of the predecessor will affect the beginning time of the successor in two ways. First, the delay will directly be added to the expected beginning time of the successor, like in PERT/CPM methodology. Second, the *lead time* computed by SDDA will decrease proportionally to the decrease in the predecessor punctuality (the development of the predecessor is considered to be less reliable), until the punctuality reaches a critical level, under which the *lead time* is equal to 0, and the dependent node will wait for the full completion of the development of the predecessor.

Criticality of Dependency The Criticality of Dependency (COD) is the critical level under which the *lead time* is equal to 0. For each developmental dependency, the parameter for COD, β_{ij}, ranges between 0 and 100, and is defined as the normalized level of punctuality of system i under which system j cannot begin its development before i is fully developed. Independently from SOD, the criticality defines the amount of risk that the manager is willing to take by applying a *lead time* to the development of systems. A high criticality means that even a small delay in the development of the predecessor will highly decrease the *lead time*, or wipe it out completely.

The model: basic formulation of Systems Developmental Dependency Analysis

Based on the parameters described in the previous section, on minimum and maximum development time, and on punctuality of each system, the output of the model is the beginning time t_B^i and the completion time t_C^i of the development of each system i.

For a root node i, that is a node without any predecessor, the beginning time is 0

$$t_B^i = 0 \tag{3.13}$$

and the completion time, depending on the punctuality, is

$$t_C^i = t_{min}^i + \left(1 - \frac{P_i}{100}\right)\left(t_{max}^i - t_{min}^i\right) \tag{3.14}$$

For a node j having at least one predecessor, SDDA first computes the time necessary for its development, t_D^j, based on its punctuality

$$t_D^j = t_{min}^j + \left(1 - \frac{P_j}{100}\right)\left(t_{max}^j - t_{min}^j\right) \tag{3.15}$$

We then calculate the beginning and completion times based on each dependency from a system i. These are the *actual* beginning and completion times of system j, if it is depending only on one system i.

If $P_i < \beta_{ij}$, system i has critical delay (left side of the plot in figure 3.22), therefore the beginning time of j based on its dependency on i, ${}^{i}t_B^j$, is equal to the completion time of i.

$$ {}^{i}t_B^j = t_C^i \tag{3.16}$$

otherwise, the beginning time of j based on its dependency on i is computed as

$$ {}^{i}t_B^j = t_C^i - t_{min}^j \left(1 - \alpha_{ij}\right) \frac{\left(P_i - \beta_{ij}\right)}{\left(100 - \beta_{ij}\right)} \tag{3.17}$$

In equation 3.17, the term that is subtracted from the completion time of i is the *lead time* of j. In this basic formulation of SDDA, the actual beginning time of a system j that has more than one dependency is the average of the beginning times resulting from each dependency. This prevents a single predecessor from critically influence the beginning time.

$$ t_B^j = \frac{1}{n} \sum_{k=1}^{n} {}^{k}t_B^j \tag{3.18}$$

The completion time of j based on its dependency on i is

$$ {}^{i}t_C^j = \max\left(t_B^j + t_D^j, t_C^i + \alpha_{ij} t_{min}^j\right) \tag{3.19}$$

The first term in the brackets is the sum of beginning time and development time, therefore it is the completion time that j would have, not accounting for the dependencies. However, the strength of each dependency does not only affect the *lead time*, but also gives a measure of the fraction of development time of j that depends on the full development of i. The second term in the bracket accounts for this dependency factor, stating that system j cannot be completed before a certain amount of time elapses after the completion of node i.

The actual completion time of system j is the maximum of the completion times given by each dependency.

$$ t_C^j = \max_n {}^{n}t_C^j \tag{3.20}$$

Computation of the beginning and completion time for each node results in a complete schedule of the development of the complex system or SoS, showing the effect of partial development dependency on the development time.

The model: conservative formulation of Systems Developmental Dependency Analysis

In the basic model of SDDA, the actual beginning time of development of a system j is computed as the average of the beginning times resulting from each dependency that system j has on other systems. If the completion time of the predecessors of j spans a large range, this choice could result in an excessive amount of *lead time*, with consequent increase in cost. To avoid this effect, a more conservative formulation can be used. In this model, called *SDDAmax* in the plots, the beginning time is the maximum of the beginning times resulting from each dependency

$$t_B^j = \max_n {}^n t_B^j \tag{3.21}$$

In the conservative model, equation 3.21 is used instead of equation 3.18.

Deterministic Analysis

In deterministic analysis, SDDA evaluates a single instance of the developmental dependencies. Given the parameters of the dependencies, and minimum and maximum development times, a single value of punctuality of each system is used to compute the resulting beginning and completion times of the development of each system. This kind of analysis is useful if the user is interested in the impact of the timeliness of a specific system on the overall development of the complex system. In this case, the user may assign values ranging between 0 and 100 to the punctuality of the system under consideration. The user may then assess the resulting impact on schedule through the analysis of beginning and completion times, listed in tables or plotted in graphic format. Another example may involve delays in several systems,

to evaluate the combined effect of multiple postponements. With deterministic analysis, the user can identify the most critical nodes under specified conditions, i.e. the nodes that most affect the development schedule. The user can compare different architectures, based on their response to delays. It must be noted that the results of this analysis also depend on the output of interest: for example, a manager might be interested in intermediate deadlines, completion time of development of intermediate systems, partial development of certain systems, delay absorption in case of low reliability. In SDDA model, delay can be partially absorbed even on the critical path, due to the partial developmental independence of the systems. Stochastic analysis better characterizes the overall impact of delays on schedule, and on the risk associated with unreliable systems in terms of punctuality. Figure 3.23 shows a Gantt chart [116] for the simple network from figure 3.21, comparing results from SDDA, SDDAmax, and PERT when all nodes have punctuality equal to 100. The matrices of strength and criticalities of the dependencies are

$$SOD = \begin{bmatrix} 0 & 0 & 0.6 \\ 0 & 0 & 0.75 \\ 0 & 0 & 0 \end{bmatrix} \qquad COD = \begin{bmatrix} 0 & 0 & 25 \\ 0 & 0 & 40 \\ 0 & 0 & 0 \end{bmatrix}$$

The minimum and maximum development time are (in weeks)

$$t_{min} = \begin{bmatrix} 10 \\ 12 \\ 10 \end{bmatrix} \qquad t_{max} = \begin{bmatrix} 16 \\ 15 \\ 14 \end{bmatrix}$$

From figure 3.23, it can be noticed that in PERT analysis node *3* must wait until its two predecessor are fully developed. SDDA and SDDAmax exhibit a *lead time* for the development of node *3*. Due to the high strength and criticality of the dependency of node *3* on node *2*, the *lead time* is small. SDDAmax is more conservative than the basic SDDA model for what concerns the risk of early development of node *3*.

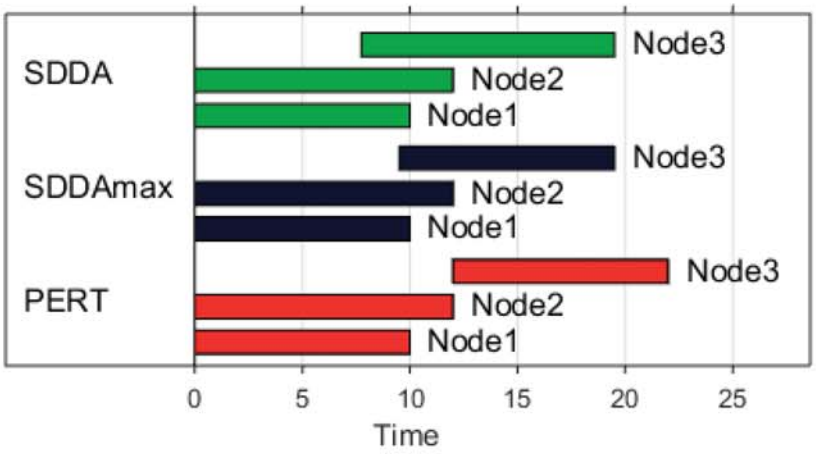


Figure 3.23. Gantt chart showing the schedule of development of the simple 3-node network shown in figure 3.21, according to the SDDA basic model, the SDDAmax model, and PERT analysis.

Table 3.14 shows the results of various instances of delays in the simple 3-node network from figure 3.21. The user is in this case interested in the completion time of the whole complex system, that is the completion time of Node 3.

Table 3.14.: Results of SDDA analysis of a simple 3-node network

Punctuality of the systems	**Model**	t_C^3 **(weeks)**	**Maximum delay in single systems (weeks)**	**Overall delay with respect to baseline (weeks)**
$\begin{bmatrix} 100 & 100 & 100 \end{bmatrix}$	SDDA	19.5	0	0
	SDDAmax	19.5	0	0
	PERT	22	0	0
$\begin{bmatrix} 75 & 100 & 100 \end{bmatrix}$	SDDA	19.5	1.5	0
	SDDAmax	19.5	1.5	0
	PERT	22	1.5	0

continued on next page

Table 3.14.: *continued*

Punctuality of the systems	**Model**	t_C^3 **(weeks)**	**Maximum delay in single systems (weeks)**	**Overall delay with respect to baseline (weeks)**
$\begin{bmatrix} 25 & 100 & 100 \end{bmatrix}$	SDDA	22	4.5	2.5
	SDDAmax	24.5	4.5	5
	PERT	24.5	4.5	2.5
$\begin{bmatrix} 100 & 75 & 100 \end{bmatrix}$	SDDA	20.25	1.6	0.75
	SDDAmax	21.29	0.75	1.79
	PERT	22.75	0.75	0.75
$\begin{bmatrix} 100 & 25 & 100 \end{bmatrix}$	SDDA	21.75	2.25	2.25
	SDDAmax	24.25	2.25	4.75
	PERT	24.25	2.25	2.25
$\begin{bmatrix} 100 & 100 & 75 \end{bmatrix}$	SDDA	19.5	1.75	0
	SDDAmax	20.5	1	1
	PERT	23	1	1
$\begin{bmatrix} 100 & 100 & 25 \end{bmatrix}$	SDDA	20.75	3	1.25
	SDDAmax	22.5	3	3
	PERT	25	3	3
$\begin{bmatrix} 50 & 50 & 100 \end{bmatrix}$	SDDA	22.38	3	2.88
	SDDAmax	23.08	3	3.58
	PERT	23.5	3	1.5
$\begin{bmatrix} 50 & 100 & 50 \end{bmatrix}$	SDDA	22.58	3	3.08
	SDDAmax	23.67	3	4.17
	PERT	25	3	3
	SDDA	21.54	2	2.04

continued on next page

Table 3.14.: *continued*

Punctuality of the systems	**Model**	t_C^3 **(weeks)**	**Maximum delay in single systems (weeks)**	**Overall delay with respect to baseline (weeks)**
$\begin{bmatrix} 100 & 50 & 50 \end{bmatrix}$	SDDAmax	25.08	2	5.58
	PERT	25.5	2	3.5
$\begin{bmatrix} 50 & 50 & 50 \end{bmatrix}$	SDDA	24.38	3	4.88
	SDDAmax	25.08	3	5.58
	PERT	25.5	3	3.5

The results listed in table 3.14 give some interesting insight:

- Under the assumption of partial dependencies, a schedule that follows the SDDA model allows for partial or total delay recovery, and results in a completion time that is lower or equal to that given by a PERT model. This possible delay recovery must be traded against the cost of longer development times of individual systems, and the increased risk due to early decision.

- A schedule that follows the SDDAmax model is more conservative than SDDA, yielding less delay recovery, but also less risk of wasting resources due to early beginning of development. The development of Node 3 is shorter in SDDAmax than in SDDA model, but the overall development time is longer.

- Node 2 is the most critical if it experiences a short delay (e.g., when punctuality is equal to [100 75 100], the delays are higher than other instances with single small decrease in punctuality). Node 1 is the most critical if it experiences a long delay (e.g., when punctuality is equal to [25 100 100], the delays are higher than other instances with single large decrease in punctuality). Delays in Node 1 are also the most critical when coupled with delays in other nodes.

Stochastic Analysis

A more accurate and complete understanding of the impact of dependencies on development can be obtained by means of a stochastic analysis with SDDA. In stochastic analysis, SDDA uses a probability density function of the punctuality, rather than a single instance. The corresponding output is a probability density function for the beginning and completion times of each system, accounting for the parameters of the model, and the overall effect of topology. The expected value of the beginning and completion times can be used as a first guideline to decisions, while the variance of these times gives insight into the risk associated with the development and delays. These outputs show patterns and features of the whole architecture. For example, given the expected distribution of punctuality, the user can compute the probability that development deadlines will be met. The user can then compare alternate architectures, identify their critical systems (i.e., systems whose delay cause the largest impact on the overall development), and identify the topological patterns that cause the observed criticality. Since SDDA is computationally inexpensive (table 3.15), Monte Carlo simulation appears to be the best choice to perform this type of analysis. The user generates a large number of instances of punctuality, based on given distributions, and then computes the beginning and completion times with SDDA.

In the examples described in this dissertation, I used the following model of uncertainty:

- The user inputs an expected level of punctuality for each system, as in SDDA deterministic analysis.
- The input level of punctuality will be the mode of a symmetric Beta probability density function (PDF), multiplied by a spreading factor. This input level might not be the mean and median of the PDF, because the tails might be cut, to respect the range of punctuality.

Table 3.15. Computational cost of SDDA analysis. Time to perform analysis of a single instance of the developmental network, with variable number of nodes and edges. Processor Intel Core i3-2350M 2.3 Ghz, 4Gb DDR3 RAM.

Number of nodes	**Average number of edges (100 runs)**	**Average time**	**Maximum time**
100	2477	$2 \cdot 10^{-3}$ s	$2.7 \cdot 10^{-3}$ s
500	62374	0.0293 s	0.0297 s
1000	249703	0.108 s	0.109 s
2000	999548	0.412 s	0.432 s

- The user inputs one of three level of uncertainty for each system (low, medium, high). The lower the uncertainty, the more reliable the assumption of the punctuality. I model this higher confidence with a lower variance of the PDF.

- The user inputs the time instant, on the development schedule, at which SDDA has to compute the expected development performance. As this time instant gets closer to a system's completion time, the uncertainty on the system punctuality will decrease (lower variance of the PDF), until the chosen time is equal or greater than the system's completion time, at which point the uncertainty on the completion time is zero.

- At this point, even if the range of the Beta function is limited (and so is the PDF resulting from the multiplication by the spreading factor, and the modified variance due to confidence and time instant), it might be partially outside the allowed range of punctuality. Hence, I cut the PDF over the range from 0 to 100, and normalize it so that its area is equal to 1.

Figure 3.24 shows the results of development analysis via the stochastic SDDA model, over 10000 Monte Carlo samples. The punctuality was generated according to

the PDF of the stochastic SDDA model, and I computed the consequent beginning and completion times of each system. The same stochastic model was also applied to PERT analysis. The matrices of strength and criticality of dependencies, minimum and maximum development times are the same as in the deterministic example of Figure 3.23. Punctuality of Nodes 1 and 3 is 100, punctuality of Node 2 is 70. Node 1 has medium uncertainty level, Node 2 has high uncertainty level, and Node 3 has low uncertainty level.

At time 0, the beginning and completion times show the largest uncertainty. At time 8 weeks, the uncertainty has decreased. Node 2 has higher uncertainty than Node 1, according to the input by the user. At time 16 weeks, two systems are fully developed, and therefore exhibit no uncertainty. Node 3 has low uncertainty, according to the input by the user. A development schedule according to the SDDA basic model shows that early completion is allowed, but the graph of the expected development at time 8 shows that there is still risk associated with the uncertainty. The user can make informed decision thanks to this methodology, choosing an appropriate beginning time, based on result of this analysis, and on the amount of accepted risk. For example, the mean of the PDF of beginning time can be used as scheduled value. The user can also calculate the probability that each system will be fully developed by given deadlines. In this example, the expected levels of punctuality did not change. Of course, these levels, as well as the levels of uncertainty, may change over time. The user can repeat the analysis later, during the development of the complex system, when further decisions are required. SDDA results suggest possible times at which to perform the analysis again, with current information. Table 3.16 lists some of the results of stochastic analysis with SDDA, SDDAmax, and PERT model, including the expected completion time, and the tenth, fiftieth, and ninetieth percentiles of the beginning time, representing more or less conservative choices. The results are computed based on the information available at time 0. The values of the parameters and inputs are the same as in the previous cases.

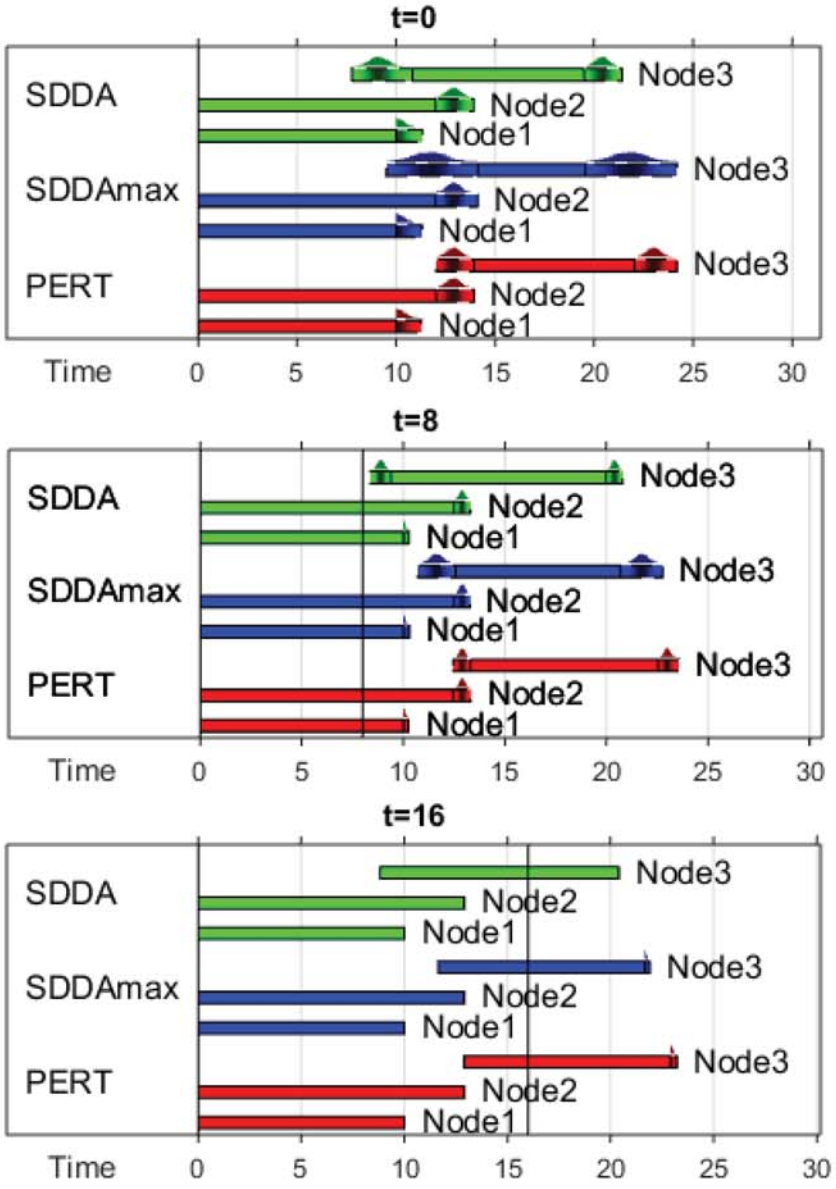


Figure 3.24. Gantt chart showing the schedule of development of the simple 3-node network shown in Figure 3.21, according to the SDDA, SDDAmax, and PERT stochastic models. The resulting PDF of beginning and completion times is shown above the corresponding bar of the Gantt chart. The darker shadows on the bars indicate the zones of higher probability. Top: analysis of expected schedule at time 0. Center: analysis of expected schedule at time 8 weeks. Bottom: analysis of expected schedule at time 16 weeks.

Table 3.16. Results of stochastic SDDA analysis of a simple 3-node network.

Baseline punctuality of the systems	**Model**	10^{th}, 50^{th}, **and** 90^{th} **percentile of** t_B^3	$E\left(t_C^i\right)$ **(weeks)**
$\begin{bmatrix} 100 & 100 & 100 \end{bmatrix}$	SDDA	$\begin{bmatrix} 7.98 & 8.33 & 8.80 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.26 & 19.76 \end{bmatrix}$
	SDDAmax	$\begin{bmatrix} 9.60 & 10.04 & 10.80 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.26 & 20.24 \end{bmatrix}$
	PERT	$\begin{bmatrix} 12.04 & 12.23 & 12.54 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.26 & 22.37 \end{bmatrix}$
$\begin{bmatrix} 70 & 100 & 100 \end{bmatrix}$	SDDA	$\begin{bmatrix} 9.21 & 9.76 & 10.34 \end{bmatrix}$	$\begin{bmatrix} 11.80 & 12.26 & 19.98 \end{bmatrix}$
	SDDAmax	$\begin{bmatrix} 9.66 & 10.16 & 10.86 \end{bmatrix}$	$\begin{bmatrix} 11.80 & 12.26 & 20.33 \end{bmatrix}$
	PERT	$\begin{bmatrix} 12.06 & 12.27 & 12.58 \end{bmatrix}$	$\begin{bmatrix} 11.80 & 12.27 & 22.40 \end{bmatrix}$
$\begin{bmatrix} 100 & 70 & 100 \end{bmatrix}$	SDDA	$\begin{bmatrix} 8.55 & 9.12 & 9.71 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.90 & 20.40 \end{bmatrix}$
	SDDAmax	$\begin{bmatrix} 10.64 & 11.65 & 12.67 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.90 & 21.76 \end{bmatrix}$
	PERT	$\begin{bmatrix} 12.48 & 12.91 & 13.33 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.91 & 23.01 \end{bmatrix}$
$\begin{bmatrix} 100 & 100 & 70 \end{bmatrix}$	SDDA	$\begin{bmatrix} 7.98 & 8.33 & 8.79 \end{bmatrix}$	$\begin{bmatrix} 10.31 & 12.26 & 19.80 \end{bmatrix}$
	SDDAmax	$\begin{bmatrix} 9.60 & 10.04 & 10.79 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.26 & 21.33 \end{bmatrix}$
	PERT	$\begin{bmatrix} 12.04 & 12.22 & 12.53 \end{bmatrix}$	$\begin{bmatrix} 10.32 & 12.26 & 23.46 \end{bmatrix}$
$\begin{bmatrix} 50 & 50 & 50 \end{bmatrix}$	SDDA	$\begin{bmatrix} 11.69 & 12.36 & 13.02 \end{bmatrix}$	$\begin{bmatrix} 13.00 & 13.50 & 24.35 \end{bmatrix}$
	SDDAmax	$\begin{bmatrix} 12.19 & 13.12 & 13.93 \end{bmatrix}$	$\begin{bmatrix} 12.99 & 13.50 & 25.09 \end{bmatrix}$
	PERT	$\begin{bmatrix} 13.15 & 13.54 & 13.94 \end{bmatrix}$	$\begin{bmatrix} 13.00 & 13.50 & 25.54 \end{bmatrix}$

As expected, PERT exhibits the latest final expected completion time $E\left(t_C^3\right)$, because no development overlapping is considered. On the other hand, the percentiles show a smaller spread in the values of beginning and completion times in PERT (Table 3.16 shows percentiles of the beginning development time of Node 3, on 10000 instances). The larger spread in the values of beginning and completion times in SDDA and SDDAmax occurs because the uncertainty in these values is not only due

to the stochastic nature of the punctuality (as it happens in PERT), but also to the *lead time* assigned to the systems, based on the parameters of the models and on the expected punctuality.

3.2.3 Source of parameters

This section is meant to provide the user with guidelines about the different possible sources of parameters to model systems behavior with SDDA. Since the main focus of this dissertation is the analytical methodology, for most applications I assessed the parameters based on expert judgment, which is faster but might be less reliable than a data-based approach. When data are not available, and the user must rely on expert opinion to model the input/output behavior of one-to-one dependencies, I recommend to perform sensitivity analysis in order to evaluate the impact of errors in the parameters, and to analyze the behavior of each architecture over a range of the most sensitive parameters.

Data-based approach

Historical data This kind of approach is based on historical data, or knowledge database. Keeping into account the meaning of the parameters, the user evaluates them based on existing information about partial parallel development of system, and punctuality confidence levels.

Input requirement-based approach An alternative data-based approach requires the user to estimate the amount of input that a receiver node requires from its feeder nodes before its own development can start. This input will suggest an appropriate SOD. Then, based on the specific problem and on the amount of risk that can be accepted, the user will evaluate the COD. Guidelines to estimate the amount of required input can be found in [82].

Expert judgment-based approach

Based on the intuitive meaning of the two parameters of SDDA model for a one-to-one dependency, an expert user can directly input appropriate parameters into the model. Figure 3.22 is useful to understand the relationship between completion time of a feeder node and beginning time of a receiver node in function of punctuality, when the parameters of the model change. In the case of expert judgment-based approach, I strongly recommend to perform an evaluation of the sensitivity of the results of SDDA analysis to changes in the parameters.

4. DEMONSTRATIVE APPLICATIONS

"Tra le sicure maniere per conseguire la verità è l'anteporre l'esperienze a qualsivoglia discorso, essendo noi sicuri che in esso, almanco copertamente, sarà contenuta la fallacia, non sendo possibile che una sensata esperienza sia contraria al vero."

"Among the safe ways to pursue truth is the putting of experience before any reasoning, we being sure that any fallacy will be contained in the latter, at least covertly, it not being possible that a sensible experience is contrary to truth."

Galileo Galilei, *Letter to Fortunio Liceti, 15 September 1640*

In this chapter, I provide examples of application of SODA and SDDA methodology to a variety of small aerospace applications. The goal is to provide a basic demonstration of how these methodologies and models can be shaped and used to address various problems. In the last section of this chapter, I use a larger version of the Naval Warfare Scenario SoS introduced in the previous chapter to show a combined application of SDDA and SODA. These examples are intended to be used as guidelines, and do not cover the whole range of applications of SODA and SDDA, nor they show all the possible modifications that can be added to the methods to expand this range. However, these applications underline how the proposed methodologies address the gaps identified in chapter 2.

4.1 Operational analysis of simple space architectures

"Listen now for the sound that forevermore separates the old from the new (introducing the beep-beep chirp transmitted by the Sputnik satellite)."

NBC radio announcer
4 October 1957, quoted in [117]

4.1.1 Earth observation system

In this section, I describe the operational dependency analysis of a high-level simple network for Earth observation. This application addresses gaps 2 and 3 from section 2.5. The Earth Observation System-of-Systems, shown in figure 4.1, is composed by two control centers (CC_1 and CC_2), one communication satellite ($ComSat$), a constellation of three observation satellites (S_1, S_2 and S_3), and a node representing the sensing capabilities that the user wants to achieve. Since this is a simple, small application meant to illustrate the use of SODA, I did not consider details about orbits of the satellites, visibility, and so on. The network is representative of the operations of the SoS during conditions in which the satellites are in view or in some relayed communication with the control centers.

In this demonstrative example, I used my expertise to determine the values of SOD, COD and IOD. However, simulations, historical data, and expert groups can be good sources of more accurate parameters. The values of SOD, COD and IOD are as follow:

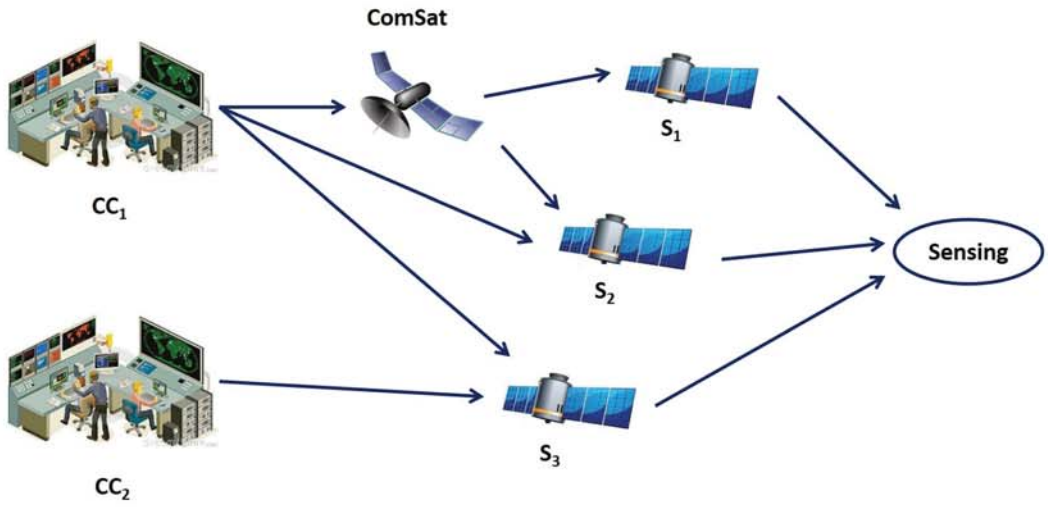


Figure 4.1. Operability dependencies of the Earth Observation SoS. CC: Control Center. ComSat: Communication Satellite. S: Observation Satellite.

$$SOD = \begin{array}{c} \\ CC_1 \\ CC_2 \\ ComSat \\ S_1 \\ S_2 \\ S_3 \\ sensing \end{array} \begin{array}{c} \begin{array}{ccccccc} CC_1 & CC_2 & ComSat & S_1 & S_2 & S_3 & sensing \end{array} \\ \begin{bmatrix} 0 & 0 & 0.3 & 0 & 0.4 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0.6 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$COD = \begin{array}{c} \\ CC_1 \\ CC_2 \\ ComSat \\ S_1 \\ S_2 \\ S_3 \\ sensing \end{array} \begin{array}{c} \begin{array}{ccccccc} CC_1 & CC_2 & ComSat & S_1 & S_2 & S_3 & sensing \end{array} \\ \left[\begin{array}{ccccccc} 0 & 0 & 70 & 0 & 60 & 60 & 0 \\ 0 & 0 & 0 & 0 & 0 & 60 & 0 \\ 0 & 0 & 0 & 90 & 70 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 40 \\ 0 & 0 & 0 & 0 & 0 & 0 & 40 \\ 0 & 0 & 0 & 0 & 0 & 0 & 40 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

$$IOD = \begin{array}{c} \\ CC_1 \\ CC_2 \\ ComSat \\ S_1 \\ S_2 \\ S_3 \\ sensing \end{array} \begin{array}{c} \begin{array}{ccccccc} CC_1 & CC_2 & ComSat & S_1 & S_2 & S_3 & sensing \end{array} \\ \left[\begin{array}{ccccccc} 0 & 0 & 100 & 0 & 50 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 25 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Deterministic Analysis

As explained in chapter 3, deterministic analysis assumes a given *working point*, i.e. a set of self-effectiveness of each system in the SoS, with the goal of assessing the effects of disruptions and reduced operability on the SoS. In this example, each *working point* represents a partial failure and consequent degraded self-effectiveness in only one system. Table 4.1 shows the corresponding operability level of each sensing satellite and of the overall sensing capability of the SoS, for different instances of degraded operability of one node.

Table 4.1. Deterministic analysis of the operability of the sensing satellites and of the overall sensing capability of the Earth Observation SoS

Disrupted system and corresponding SE	O_{S_1}	O_{S_2}	O_{S_3}	$O_{sensing}$
CC_1 (70)	94.6	93.1	95.5	94.4
CC_1 (25)	73	80.5	88.75	80.75
CC_2 (70)	100	100	89.5	96.5
CC_2 (25)	100	100	73.75	91.25
$ComSat$ (25)	68.5	94.75	100	87.75
S_1 (25)	70	100	100	90
S_2 (25)	100	47.5	100	82.5
S_1 (25)	100	100	62.5	87.5

These results show that the first control center, CC_1, is the most critical system in the architecture. Also, the overall capability is more sensitive to a partial failure in the second observation satellite, S_2, than in the other two observation satellites. Information about the criticality can be used both when deciding actions to be taken against the criticality of an individual systems (assuring that the critical system maintains high reliability), and when architecting the entire SoS (for example, deciding the number of satellites required to keep the operability above a given threshold in case of failures).

Stochastic Analysis

As an example of stochastic analysis, figure 4.2 shows the operability of sensing capability following degradation in the control centers and the communication satellite, that exhibit SE following a Beta(2,8). In this case three systems are experiencing a reduced level of SE at the same time, according to the given probability density

function. The expected value of the SE of the control centers and the communication satellite is equal to 20, while the resulting operability of the sensing capability has an expected value equal to 58. This result gives a first insight into the robustness of the architecture, i.e. its capability to maintain a high level of operability following disruptions and partial failures.

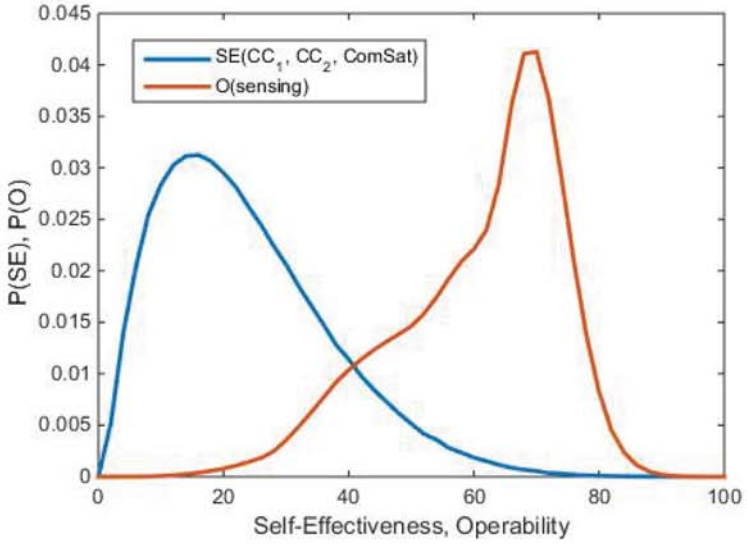


Figure 4.2. Operability of sensing capability of the Earth Observation SoS following disruptions in the control centers and the communication satellite.

Evolution of operability over time and Robustness

SODA can be used to evaluate the evolution of the operability over time. This results can be used to build better metrics for robustness, including part or all of the life cycle of the SoS. I used a simple failure model to evaluate the evolution of the distribution of SE of each system, and SODA to quantify the evolution of operabilities based on the topology of the network and the SE of the systems at each time step. This computation relates the reliability of individual systems to that of the whole SoS.

Based on these results, the user can quantify the robustness of an architecture, and use it to compare different architectures or to trade off with other measures of performance. Defining robustness as the capability of a system to keep high performance following disruptions, I propose the following metric for robustness (with stochastic analysis and continuous time):

$$Rob_{\mathcal{A}} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{t_f}\int_{t=0}^{t_f}\left(\frac{\frac{1}{m}\sum_{j\in\mathcal{I}} O_j(t,n) - \min_{N\in\mathcal{A}} SE_N(t,n)}{100 - \min_{N\in\mathcal{A}} SE_N(t,n)}\right)\right) \tag{4.1}$$

where $\mathcal{A}$ is an architecture, represented by a network of nodes and edges $G = (N, E)$ and appropriate matrices of SOD, COD, and IOD, n is the number of instances (since I am using a stochastic model and a tool that allows for fast analysis, as mentioned in chapter 3, Monte Carlo simulation is a preferred method to compute results), t_f is the final time, m is the number of nodes of interest, used to compute the *overall capability* of the architecture, $\mathcal{I}$ is the subset of nodes of interest and N are all the nodes in architecture $\mathcal{A}$. This formulation requires the lowest SE in the network to be strictly less than 100. At each time instant, the difference between the average operability of interest and the lowest SE in the network is compared to the difference between 100 and the lowest SE in the network. This is the fraction of maximum loss in a node that is recovered in the average operability of interest thanks to the operational dependencies. A robustness value of 1 means that the operability of interest is not affected by the loss, and the architecture has maximum robustness. A robustness value of 0 means that the highest disruption affects the overall operability without any recovery due to the architecture. This would happen if the node experiencing the disruption has no dependency from any other node (so that its operability is equal to its SE) and all the paths in the network from the node experiencing the disruption to the nodes of interest have maximum strength (so that the operability of every node following the disrupted node is depending only on this disrupted input). These values of robustness at each time instant are averaged over time (through an integration) and averaged over the stochastic instances. In case of discrete time steps, the integral is replaced by a sum.

It is important to underline a few considerations about this metric:

- There are many possible metrics for robustness that can be built based on the results of SODA in terms of operability of the systems. The user can decide to tailor this metric to a specific problem, or even to use a completely different metric.

- This metric is a *relative* measure of robustness, meaning that it is computed using the difference between the average operability and the minimum SE. The reason is that robustness is not meant to measure the absolute performance of the network, but how well the network is capable to absorb disruptions. Intuitively, this means that a topology having an average operability of interest equal to 80 when the minimum SE in any node is 40 is more robust than a topology having the same average operability of interest when the minimum SE in any node is 70.

- For the same reason, this metric is a ratio, i.e. the percentage of loss that is absorbed. Intuitively, this means that a topology "recovering" an amount of operability equal to 20 when the maximum loss in any node is 20 is more robust than a topology exhibiting the same "recovery" when the maximum loss in any node is 80.

- While this metric is meant to be a function of the topology only, the simple practical formulation also depends on the evolution of the SE of each node over time, and on the interval of integration. Figure 4.5 shows that the robustness of the topology at each time step and on each instance does not vary over a wide range of minimum SE. However, low levels of robustness are computed with this formulation when the maximum loss in a single node is small. This occurs because, in case of very small losses in SE, the effect of operational dependencies across the network is relatively tiny, so that the experienced recovery in operability is just a fraction of the small loss in SE.

- This metric has the objective of facilitating architecture comparison, not to identify the criticalities. Therefore, it does not distinguish which node is experiencing the maximum loss in SE. This is the cause of the variability of average operability of interest as a function of a given minimum SE (figure 4.5). Other metrics can be easily formulated which consider losses only in one node, so as to measure the robustness to failures on a specific system (figure 4.9).

- For the same reason, this metric does not distinguish between the case of single failure and multiple failures. Once again, more metrics which will distinguish robustness in case of single failure from robustness in case of unlimited number of failures can be easily derived from this baseline metric.

In this example, I simulated and analyzed 100 discrete time steps. Beginning with $SE = 100$, at each time step the control centers, communication satellite, and observation satellites had a decrease in SE due to aging equal to $0.01 + 0.04\,unif[0, 1]$, i.e. a probabilistic value between 0.01 and 0.05. Furthermore, at each time step each system could experience a minor failure, with a probability of 1% and a loss in SE equal to $3 + 3\,unif[0, 1]$, and a major failure, with a probability of 0.6% and a loss in SE equal to $30 + 20\,unif[0, 1]$. Figure 4.3 shows the evolution over time of the SE of the control centers, communication satellite, and observation satellites in one instance of their life cycle of 100 time steps.

With $n = 40000$, $m = 1$ and $\mathcal{I} = \{sensing\}$, the robustness of the Earth observation SoS under these conditions is $Rob_{EarthObs} = 0.60$. Figure 4.4 shows the histogram of the average robustness over all the instances. Figure 4.5 is a scatter plot of the average robustness versus the minimum operability over all the instances. The robustness measured in this case is relative to the aging and the minor and major failure modeled.

In case the user does not wish to introduce a model of change of SE over time, but rather just probability distribution functions for the SE of one or more systems, the following formulation of robustness can be used (again, already formulated in terms

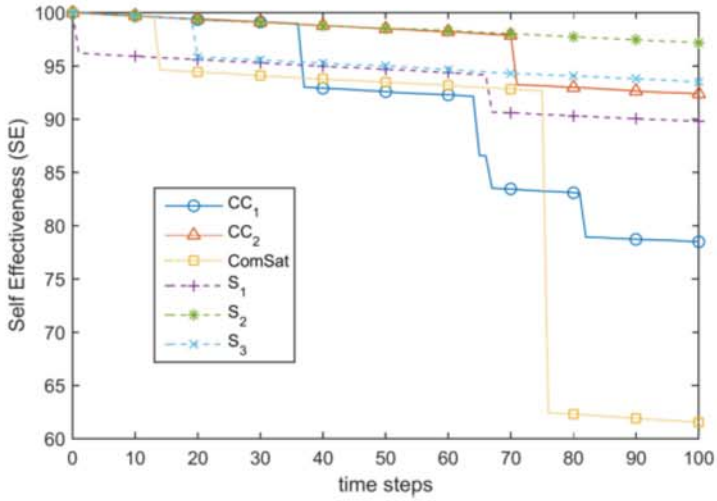


Figure 4.3. Evolution over time of systems SE in the Earth Observation SoS. Control Center 1 experiences multiple minor failures, the communication satellite experiences a minor and a major failure, the second sensing satellite experiences only decrease in SE due to aging.

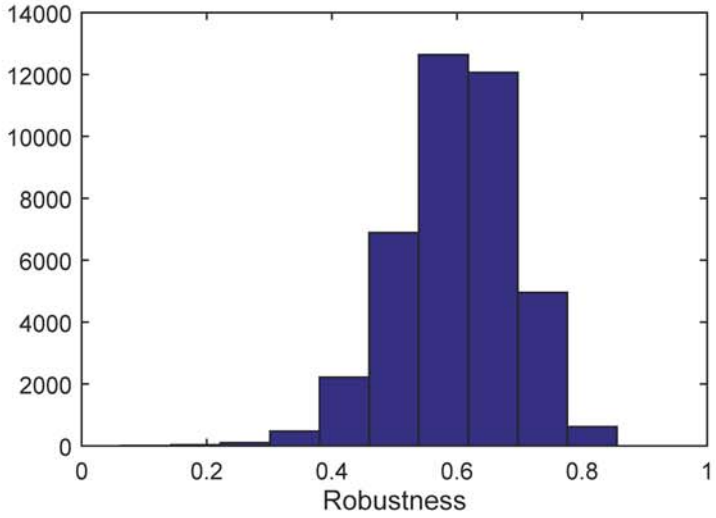


Figure 4.4. Histogram of robustness of the Earth Observation SoS with aging and failures, over 40000 instances.

of Monte Carlo simulation, for practical reasons. However, this is just the expected value of the ratio in the formula):

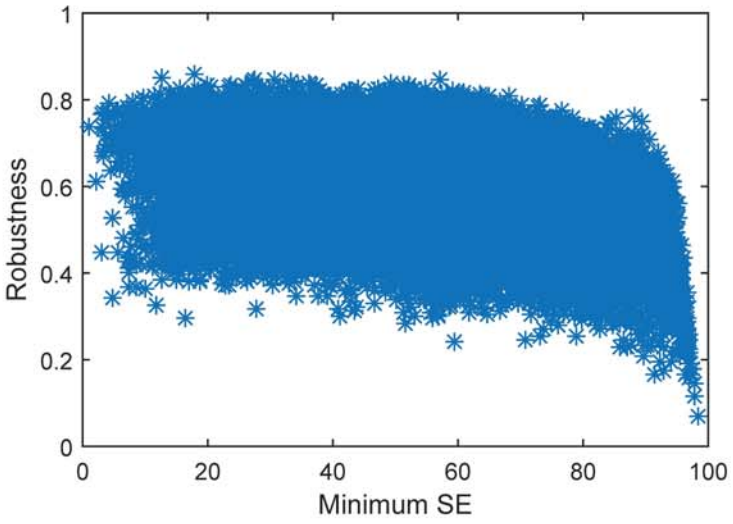


Figure 4.5. Scatter plot of robustness of the Earth Observation SoS. Each star plots the average robustness versus the average lowest SE over time in one instance.

$$Rob_{\mathcal{A}} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\frac{1}{m}\sum_{j\in\mathcal{I}} O_j(n) - \min_{N\in\mathcal{A}} SE_N(n)}{100 - \min_{N\in\mathcal{A}} SE_N(n)}\right) \tag{4.2}$$

With this formulation, using a Beta(8,2) density function for the SE of the control centers, communication satellite, and sensing satellites, $n = 100000$, $m = 1$ and $\mathcal{I} = \{sensing\}$, the robustness of the Earth observation SoS is $Rob_{EarthObs} = 0.44$. This lower value is due to the fact that the chosen model for the SE causes multiple failures to be more frequent, and the minimum SE is often accompanied by major failures in other systems at the same time.

Figure 4.6 shows the histogram of the average robustness over all the instances. Figure 4.7 is a scatter plot of the robustness versus the minimum operability over all the instances. The robustness measured in this case is relative to possible multiple failures, and SE modeled with Beta(8,2) for all systems simultaneously.

Using the formulation in equation 4.2, the user can also quantify the robustness to failures in a single system. I used again a Beta(8,2) density function for the SE of the control centers, communication satellite, and sensing satellites (only one system

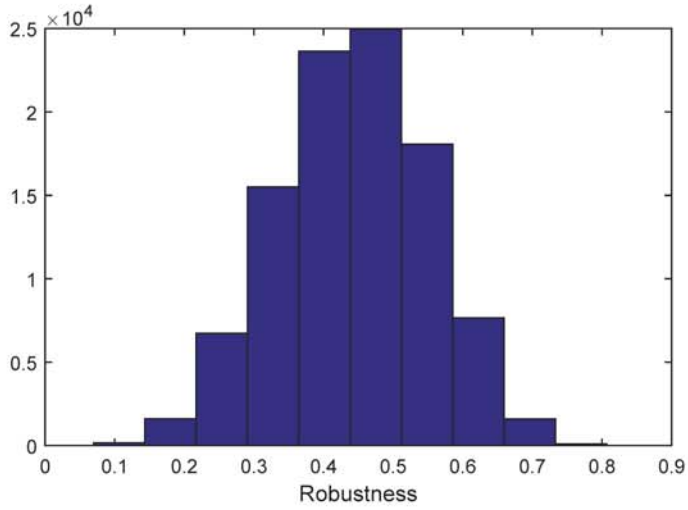


Figure 4.6. Histogram of robustness of the Earth Observation SoS with multiple failures (SE modeled with Beta(8,2), over 100000 instances).

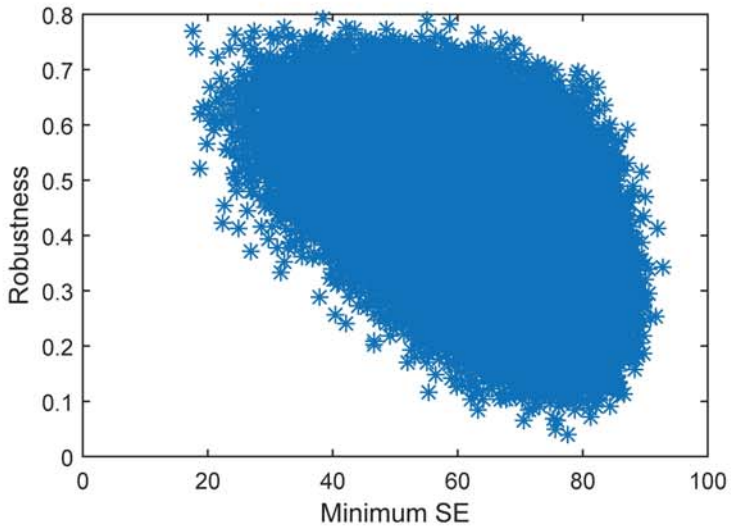


Figure 4.7. Scatter plot of robustness of the Earth Observation SoS. Each star plots the robustness versus the lowest SE in one instance.

at a time has SE according to this function, while the others have maximum SE), $n = 20000$ per each failed system, $m = 1$ and $\mathcal{I} = \{sensing\}$. The robustness of the Earth observation SoS to failures in each system N, $Rob^N_{EarthObs}$, is listed in table 4.2.

Table 4.2. Robustness of the Earth observation SoS to failures in a single system.

Disrupted system (modeled with Beta(8,2))	**Robustness** $Rob^{N}_{EarthObs}$
CC_1	$Rob^{CC_1}_{EarthObs} = 0.745$
CC_2	$Rob^{CC_2}_{EarthObs} = 0.882$
$ComSat$	$Rob^{ComSat}_{EarthObs} = 0.837$
S_1	$Rob^{S_1}_{EarthObs} = 0.867$
S_2	$Rob^{S_2}_{EarthObs} = 0.767$
S_1	$Rob^{S_3}_{EarthObs} = 0.833$

The architecture is very robust to single failures, meaning that the redundancies and topology make it capable to maintain high operability following failures in one system. The results also confirm that the architecture is less robust in case of failures in the first control center and in the second sensing satellite, analogously to what observed when studying the criticality with deterministic analysis. The average robustness to single failures is $Rob_{EarthObs} = 0.82$.

Figure 4.8 shows the histogram of the robustness over all the instances. Figure 4.9 is a scatter plot of the robustness for single failures versus the minimum operability over all the instances. As expected, since the failure occurs in only one system, there is no variability of robustness given the minimum SE, which is the SE of the disrupted system. The robustness measured in this case is relative to single failure, and disrupted SE modeled with Beta(8,2). Interesting information comes from this plot. The two most critical systems, i.e. the first control center and the second sensing satellite, exhibit different behavior. Failures in the second sensing satellite cause a relatively low robustness of the architecture which is constant for all amount of lost capability in the satellite. Instead, minor failures in the first control center result in a slightly higher robustness with respect to failure in the second sensing satellites. As the SE of the control center decreases, however, the architecture result less robust, i.e.

less capable to maintain high operability, and more prone to propagate the negative impact of the failure in the control center to the overall capability. This indicates that, for low level of SE in the first control center, one or more dependencies along the path from the control center to the node representing the sensing capability enter the COD zone (figure 3.4), having therefore less recoverable impact, which is quantified as a reduction in the robustness of the architecture.

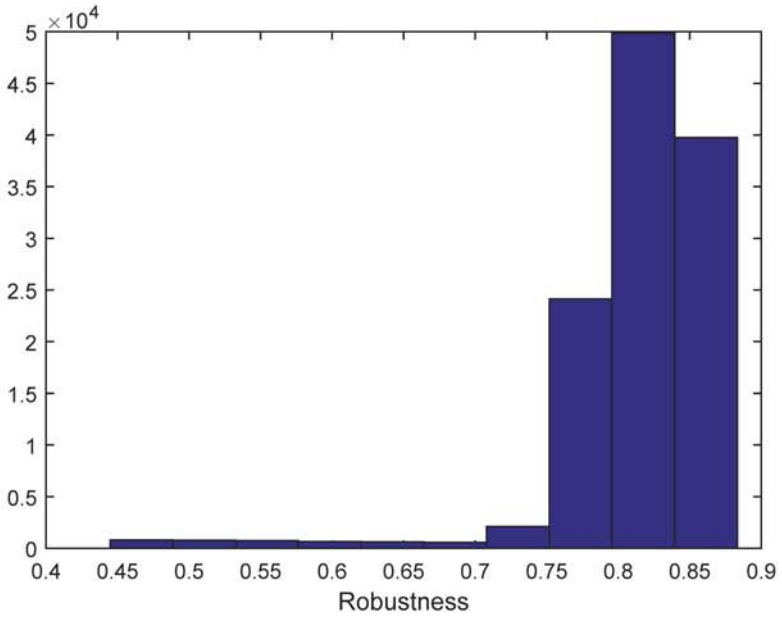


Figure 4.8. Histogram of robustness of the Earth Observation SoS with single failures (SE modeled with Beta(8,2), over 120000 instances).

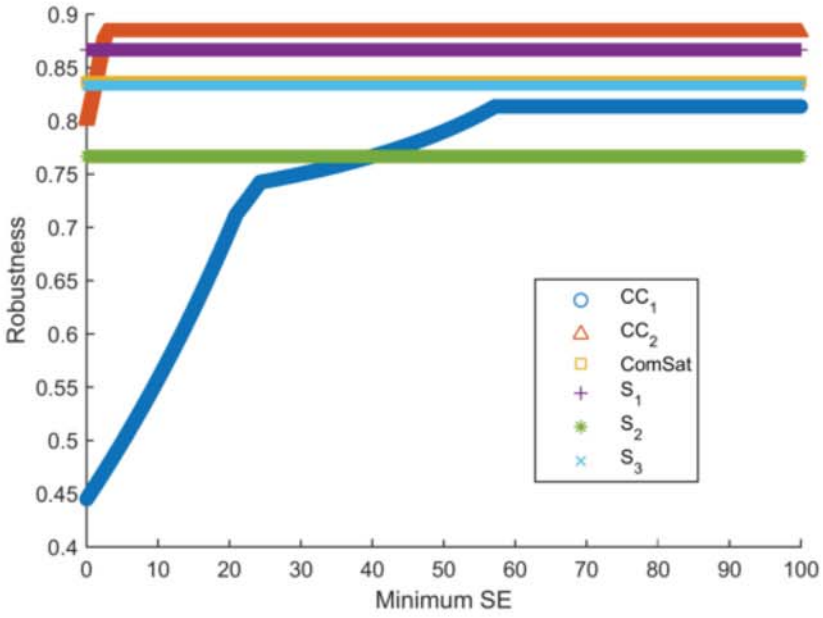


Figure 4.9. Scatter plot of robustness of the Earth Observation SoS with single failures. Each star plots the robustness versus the lowest SE in one instance.

Flexibility and Resilience

More complex analysis, yielding more information, can be added to the basic SODA analysis. For example, the user can keep in consideration the flexibility of an architecture. This means that some systems can modify their dependencies if some events occur, and the architecture can change overtime to redistribute excess capabilities, or re-task systems following the loss of desired capabilities. The alternate architecture must keep the same nodes representing capabilities of interest, but may have a different set of nodes and edges, and/or different modeling parameters, with respect to the baseline architecture.

Thanks to flexibility, an architecture can exhibit resilience. Resilience has many different definitions in literature [26], but it is mostly described as the capability to react to failures and degradation in order to recover operability, at least partially. Using the same concepts that brought to the formulation of a quantitative metric for

the robustness of an architecture, I developed the following formulation of a metric for resilience of an architecture:

$$Res_{\mathcal{A}*} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{t_f}\int_{t=0}^{t_f}\left(\frac{\frac{1}{m}\sum_{j\in\mathcal{I}} O*_j\,(t,n) - \min_{N\in\mathcal{A}} SE_N(t,n)}{100 - \min_{N\in\mathcal{A}} SE_N(t,n)}\right)\right) \tag{4.3}$$

where $\mathcal{A}$ is the baseline architecture, represented by a network of nodes and edges $G = (N, E)$ and appropriate matrices of SOD, COD, and IOD, n is the number of instances (since I am using a stochastic model and a tool that allows for fast analysis, as mentioned in chapter 3, Monte Carlo simulation is a preferred method to compute results), t_f is the final time, m is the number of nodes of interest, used to compute the *overall capability* of the architecture, $\mathcal{I}$ is the subset of nodes of interest and N are all the nodes in architecture $\mathcal{A}$. In this case, $O*_j$ indicates the operability of node j in the baseline architecture $\mathcal{A}$ if the conditions are nominal, and in the alternate architecture $\mathcal{A}*$ is the conditions require to exploit the flexibility of the baseline architecture and switch to the alternate. This formulation requires the lowest SE in the network to be strictly less than 100.

At each time instant, the difference between the average operability of interest and the lowest SE in the network is compared to the difference between 100 and the lowest SE in the network. Differently from the metrics for robustness, the operability of interest can be higher due to the flexibility of the architecture. Therefore, the metric evaluates the fraction of maximum loss in a node that is recovered in the average operability thanks to both the operational dependencies and the flexibility of the architecture. A resilience value of 1 means that the operability of interest is fully recovered following the loss, and the architecture has maximum resilience. A resilience value of 0 means that the highest disruption affects the overall operability without any recovery due to the architecture or to the flexibility. These values of resilience at each time instant are averaged over time (through an integration) and averaged over the stochastic instances. In case of discrete time steps, the integral is replaced by a sum.

Similarly to what I listed about robustness, there are a few important considerations about this resilience metric:

- There are many possible metrics for resilience that can be built based on the results of SODA in terms of operability of the systems and on possible flexibility or stakeholder decisions. The user can decide to tailor this metric to a specific problem, or even to use a completely different metric.

- This metric, similarly to the metric for robustness, is a *relative* measure of resilience, meaning that it is computed using the difference between the average operability with flexibility and the minimum SE in the baseline architecture. The reason is that resilience is not meant to measure the absolute performance of the network, but how well the network is capable to recover after disruptions. Intuitively, this means that a flexible topology reaching an average operability of interest equal to 80 when the minimum SE in any node of the baseline architecture is 40 is more resilient than a topology reaching the same average operability of interest when the minimum SE in any node in the baseline architecture is 70.

- For the same reason, this metric is a ratio, i.e. the percentage of loss that is recovered. Intuitively, this means that a flexible topology recovering an amount of operability equal to 20 when the maximum loss in any node in the baseline architecture is 20 is more resilient than a flexible topology exhibiting the same recovery when the maximum loss in any node in the baseline architecture is 80.

- This metric is a function of the topology and of the flexibility. Different rules of available flexibility, including redundancy, added and removed systems and edges result in general in different values of resilience. For this reason, the metric is specific to a given alternate architecture $\mathcal{A}*$. SODA and the resilience metric can be used together with existing approaches, to add considerations about complexity, ease of change, and cost of change, for example the methodologies described by Tamaskar et al. [118], Lafleur [86], Beesemyer et al. [25] and Davendralingam and DeLaurentis [38].

- This simple practical formulation of the metric is also depending on the evolution of the SE of each node over time, and on the interval of integration.

- The alternate architecture comes into play only when needed, so the user must set also requirements for the onset of the alternate architecture, for example a minimum acceptable threshold of capability of interest from the baseline architecture. This means that, for instances and time intervals when the baseline architecture is in use, the value of resilience is equal to that of robustness. However, when the alternate architecture is in use, the capability of interest is expected to be higher than with the baseline architecture, resulting in an average resilience higher than the average robustness.

- This metric has the objective of facilitating architecture comparison and evaluation of the available flexibility, not to identify the criticalities. Therefore, it does not distinguish which node is experiencing the maximum loss in SE. Other metrics can be easily formulated which consider losses only in one node, so as to measure the resilience to failures on a specific system.

- For the same reason, this metric does not distinguish between the case of single failure and multiple failures. Once again, more metrics which will distinguish resilience in case of single failure from resilience in case of unlimited number of failures can be easily derived from this baseline metric.

For example, suppose that the control centers can perform part of each other's tasks, in case of degradation of the overall behavior. This means that two new edges become part of the architecture, connecting the second control center to the communication satellite and to second sensing satellite. The criticality of the first control center will be lower. Even if this simulation, using the same aging model used to compute robustness, does not consider which is the disrupted system, and just "blindly" applies the alternate architecture when the sensing capability becomes lower than 70, there is still a slight improvement brought by the alternate architecture, with resilience equal to 0.61, and average sensing capability increasing from 83.6 to

84.8. Figure 4.10 shows the histogram of the average resilience over all the instances. Figure 4.11 is a scatter plot of the average resilience versus the minimum operability over all the instances.

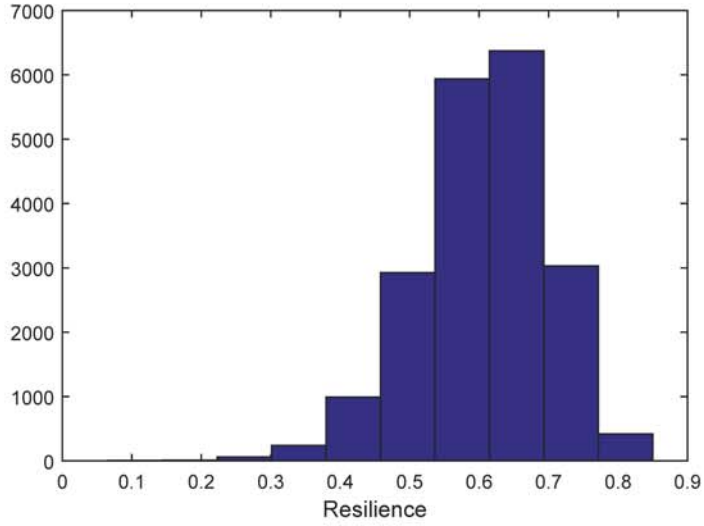


Figure 4.10. Histogram of resilience of the Earth Observation SoS with aging and failures, over 20000 instances.

If the alternate architecture is implemented only in case of disruptions in the first control center, the resulting resilience is 0.63 and the average sensing capability 85.2. Figure 4.12 shows the histogram of the average resilience over all the instances in this case. Figure 4.13 is a scatter plot of the average resilience versus the minimum operability over all the instances in this case.

Similarly to what done for the metric of robustness, in case the user does not wish to introduce a model of change of SE over time, but rather just probability distribution functions for the SE of one or more systems, the following formulation of resilience can be used (again, already formulated in terms of Monte Carlo simulation, for practical reasons):

$$Res_{\mathcal{A}*} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\frac{1}{m}\sum_{j\in\mathcal{I}} O*_j(n) - \min_{N\in\mathcal{A}} SE_N(n)}{100 - \min_{N\in\mathcal{A}} SE_N(n)}\right) \tag{4.4}$$

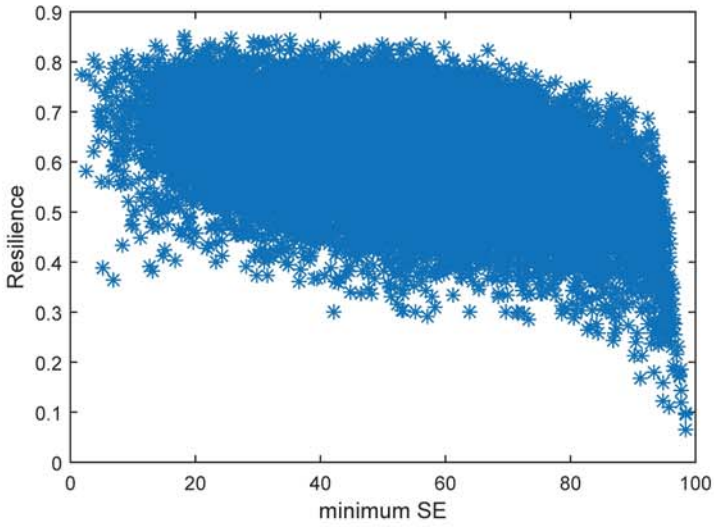


Figure 4.11. Scatter plot of resilience of the Earth Observation SoS. Each star plots the average resilience versus the average lowest SE over time in one instance.

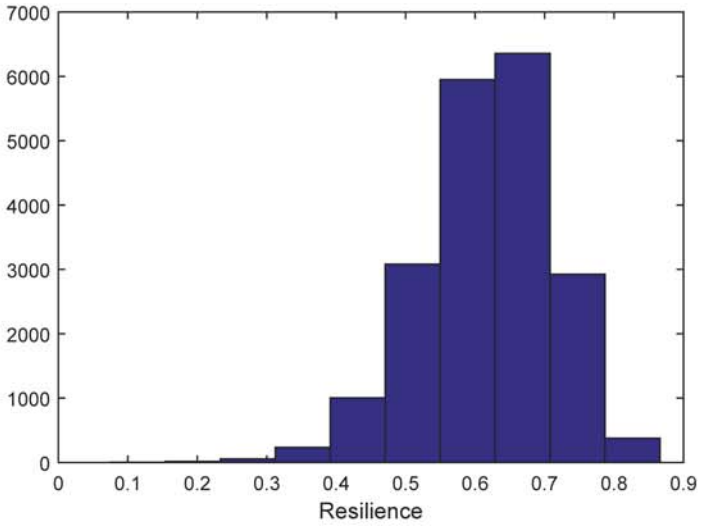


Figure 4.12. Histogram of resilience of the Earth Observation SoS with aging and failures, over 20000 instances, when flexibility is added following failures in the first control center.

With this formulation, using a Beta(8,2) density function for the SE of the control centers, communication satellite, and sensing satellites, $n = 100000$, $m = 1$ and

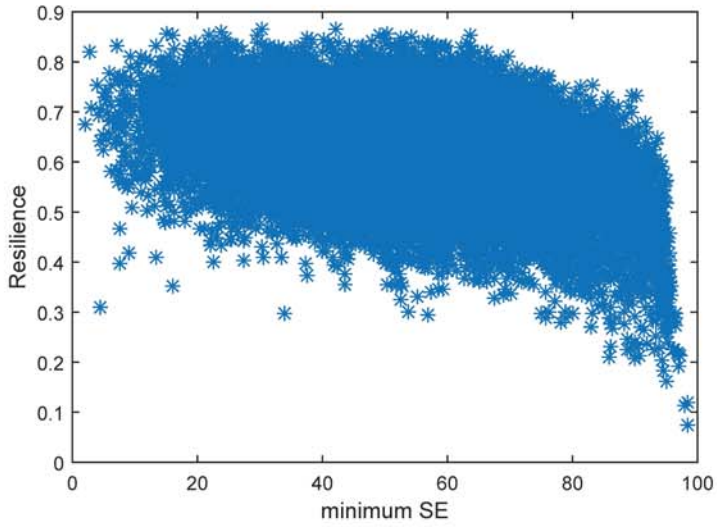


Figure 4.13. Scatter plot of resilience of the Earth Observation SoS, when flexibility is added following failures in the first control center.. Each star plots the average resilience versus the average lowest SE over time in one instance.

$\mathcal{I} = \{sensing\}$, the resilience of the Earth observation SoS when case the alternate architecture comes into play in case of disruption in the first control center is $Res_{EarthObs} = 0.46$. This lower value is due to the fact that the chosen model for the SE causes multiple failures to be more frequent, and the minimum SE is often accompanied by major failures in other systems at the same time.

Figure 4.14 shows the histogram of the average resilience over all the instances. Figure 4.15 is a scatter plot of the resilience versus the minimum operability over all the instances. The resilience measured in this case is relative to possible multiple failures, and SE modeled with Beta(8,2) for all systems simultaneously.

Using the formulation in equation 4.4, the user can also quantify the resilience to failures in a single system. This analysis yields the most interesting results. Since the alternate architecture is supporting the first control center with the second one, a relatively large increase from the robustness of architecture $\mathcal{A}$ to the resilience obtained thanks to the alternate architecture $\mathcal{A}*$ should occur when a single failure

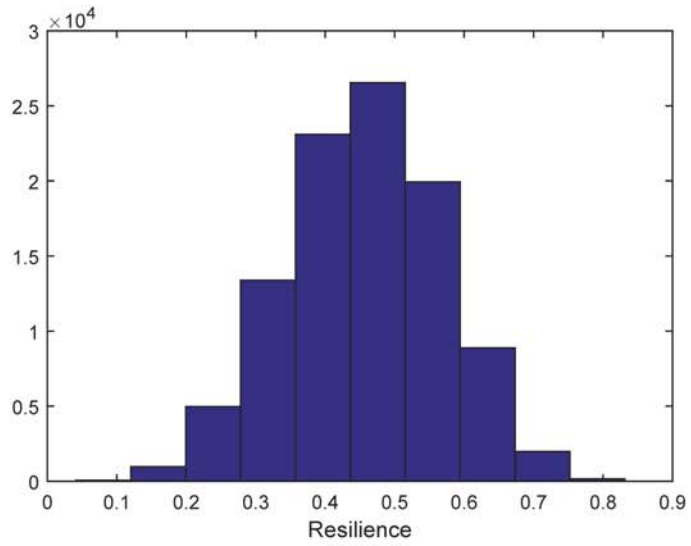


Figure 4.14. Histogram of resilience of the Earth Observation SoS with multiple failures (SE modeled with Beta(8,2), over 100000 instances).

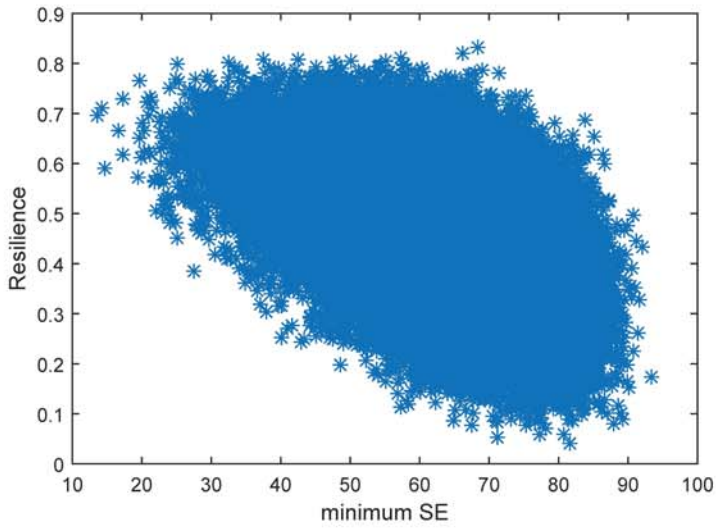


Figure 4.15. Scatter plot of resilience of the Earth Observation SoS. Each star plots the resilience versus the lowest SE in one instance.

disrupts the first control center. I used a Beta(8,2) density function for the SE of the disrupted, $n = 20000$, $m = 1$ and $\mathcal{I} = \{sensing\}$, and an accepted threshold of the operability of sensing equal to 85. Below this level, the alternate architecture sets on.

The resilience of the Earth observation SoS to failures in each system N, $Res^{N}_{EarthObs}$, is listed in table 4.3.

Table 4.3. Robustness and resilience of the Earth observation SoS to failures in a single system.

Disrupted system (modeled with Beta(8,2)	**Robustness** $Rob^{N}_{EarthObs}$	**Resilience** $Res^{N}_{EarthObs}$
CC_1	$Rob^{CC_1}_{EarthObs} = 0.745$	$Res^{CC_1}_{EarthObs} = 0.841$
CC_2	$Rob^{CC_2}_{EarthObs} = 0.882$	$Res^{CC_2}_{EarthObs} = 0.878$
$ComSat$	$Rob^{ComSat}_{EarthObs} = 0.837$	$Res^{ComSat}_{EarthObs} = 0.839$
S_1	$Rob^{S_1}_{EarthObs} = 0.867$	$Res^{S_1}_{EarthObs} = 0.867$
S_2	$Rob^{S_2}_{EarthObs} = 0.767$	$Res^{S_2}_{EarthObs} = 0.773$
S_1	$Rob^{S_3}_{EarthObs} = 0.833$	$Res^{S_3}_{EarthObs} = 0.830$

This architecture with the given rules of flexibility exhibit high resilience to single failures, meaning that the high robustness is supported by the small changes in the topology resulting in the alternate architecture, and the SoS is capable to recover operability following failures. A quick, yet wrong consideration might lead to think that the alternate architecture is generally preferable to the baseline architecture. This is not always the case. For example, the second control center must sacrifice some of its capability in order to support the first control center in the alternate architecture. For this reason, in case of failure in the second control center, the alternate architecture is less capable to recover operability than the baseline architecture. Table 4.3 shows how the alternate architecture performs much better in case of failures in the most critical systems, i.e. the first control center and the second communication satellite, suggesting to switch from the baseline architecture to the alternate architecture in case these kind of failures occur. The average resilience to single failures, using the alternate architecture in any case where the operability of sensing is below the threshold, is $Res_{EarthObs} = 0.84$.

Figure 4.16 shows the histogram of the resilience over all the instances. Figure 4.17 is a scatter plot of the robustness and resilience for single failures in the two most critical systems versus the minimum operability over all the instances. When the alternate architecture comes into play, the SoS is more capable to recover from disruptions, resulting in a value of resilience higher than the corresponding robustness. Similarly to what described for the robustness, as the SE of the control center decreases the architecture result less robust, i.e. less capable to maintain high operability, and more prone to propagate the negative impact of the failure in the control center to the overall capability. Even the plot of the resilience to failures in the control center shows the onset of points where one or more dependencies along the path from the control center to the node representing the sensing capability enter the COD zone (figure 3.4). This results in a less recoverable impact, which is quantified as a reduction in the resilience of the architecture.

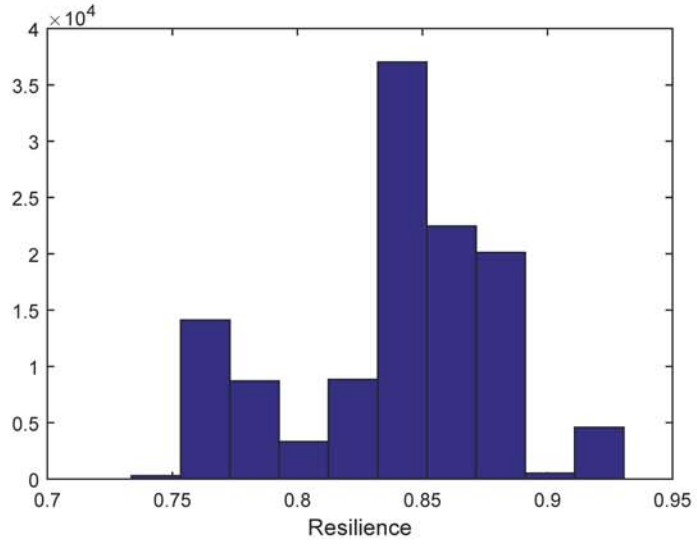


Figure 4.16. Histogram of resilience of the Earth Observation SoS with single failures (SE modeled with Beta(8,2), over 40000 instances).

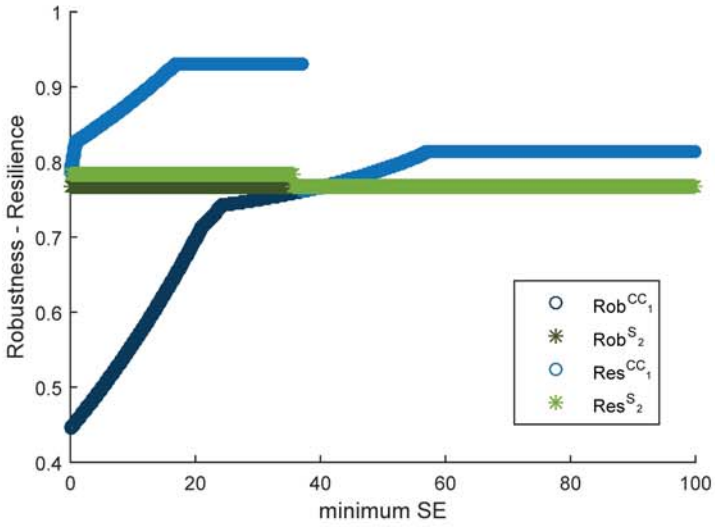


Figure 4.17. Scatter plot of robustness and resilience of the Earth Observation SoS with single failures in the first control center and in the second sensing satellite. Each star plots the robustness versus the lowest SE in one instance.

4.1.2 On-orbit satellite servicing System-of-Systems

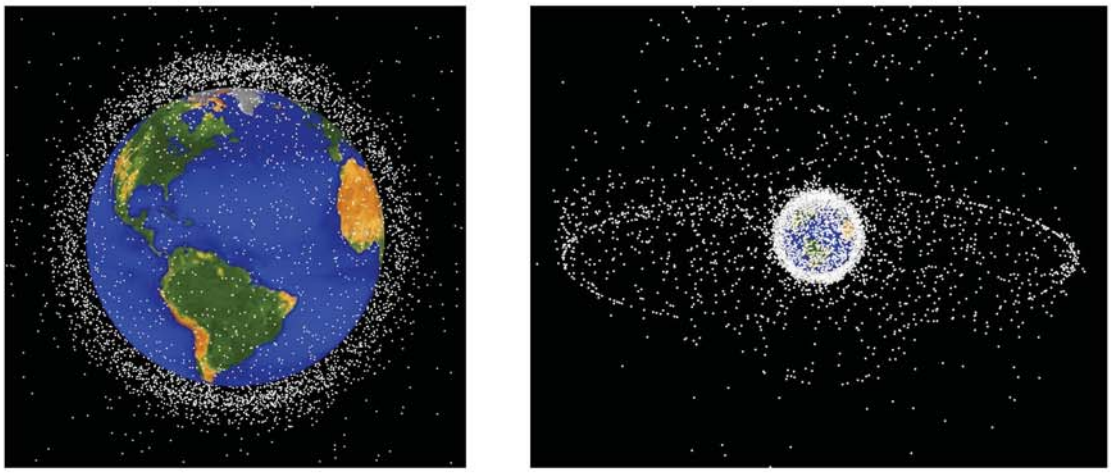
In this section, I describe the use of SODA on a large-scale, hierarchical SoS. This application addresses gaps 1, 2 and 3 from section 2.5. The SoS under consideration consists of satellite missions, including a required capability and either one satellite or a constellation of two to seven satellites, and by servicing satellites. Servicing satellites in orbit yields two advantages. First, it reduces the cost of space missions, because in case of failures a satellite might be repaired and restored to at least partial operability, instead than be discarded and replaced with an entirely new satellite. In 1993, the concept of on-orbit servicing had its most famous exposure when the first servicing mission to the Hubble Space Telescope saved a project whose cost at the launch was 1.5 billion dollars, and risked to fail due to an excessive flatness in the primary mirror surface of just 2.2 μm (figure 4.18).

Second, on-orbit servicing can reduce the production of space debris. As of January 2016, while the catalog of operating satellites by the Union of Concerned Sci-

Figure 4.18. Astronauts Extra-Vehicular Activity during the first Hubble Space Telescope servicing mission.

entists has 1381 entries, the U.S. Space Surveillance Network estimates about 3000 intact artificial satellites still orbiting Earth, and keeps a catalog of more than 21000 pieces of space debris larger than 10 *cm* (figure 4.19). Deorbiting maneuvers of satellites that reach their end-of-life are not always provided for, or in some cases cannot be executed due to failures. Furthermore, satellites may need simple servicing operations, like refueling, deployment of entangled structures, or recovery from failed orbit insertions. Therefore, on-orbit servicing is suitable to lengthen the life of old or disrupted satellites, as well as to reduce the production of more space debris.

On-orbit satellite servicing operations are defined as the process of improving a space-based capability through a combination of on-orbit activities that may include inspection, rendezvous and docking, refueling, and value-added modifications to a satellite's position, orientation, and operational status [119]. Historic surveys show that many of the past satellite servicing missions have been based on the use of the Space Shuttle, and have been performed by astronauts. Later, automated servicing



(a) Catalogued debris in Low Earth Orbit

(b) Catalogued debris in High, Medium, and Low Earth Orbit. The "ring" corresponds to Geostationary Earth Orbit

Figure 4.19. Debris in Earth orbit

missions have been developed and tested: in 1997, the Japanese Agency for Space Development (NASDA, now merged into the Japanese Aerospace Exploration Agency, JAXA) launched the Engineering Test Satellite VII (ETS-VII), an unmanned spacecraft that successfully performed rendezvous and docking with a non-cooperative target, Orbital Replacement Unit (ORU) exchange, deployment of space structures, capture of a target, and other robotic experiments [120]. A whole decade elapsed before NASA, in 2007, launched the Orbital Express mission. In this experimental mission two satellites, ASTRO and NEXTSat performed robotic experiments similar to those executed by ETS-VII, but with increased complexity. In addition, the satellites successfully completed refueling, and orbit change maneuvers. Both the ETS-VII experiment and the Orbital Express mission, as well as research publications, were focused on the complex robotic operations, with the main focus being the dynamics of the rendezvous, capture, and robotic arms in space [121–123]. To deal with the complexity of on-orbit servicing, the experimental satellites were designed taking into account serviceability and ease of access of replaceable parts. For the same reason,

and since complexity prevented the commercial development of on-orbit servicing, Neema et al. [33] proposed the concept of designing modular satellites with easily accessible and replaceable modules, to allow rejuvenation of satellites experiencing a failure, as a means to mitigate the issue of debris in space.

In 2010, NASA published a report about on-orbit satellite servicing operations [124]. This report addresses two popular myths about on-orbit servicing. The first one is "servicing is costly". Sullivan [125] studied the economical feasibility of telerobotic on-orbit servicing, finding that on-orbit servicing is commercially valuable. Long et al. [119] show how new concepts of design, with less redundancy, result in cheaper satellite, and the increased risk of failure is mitigated by the possibility of receiving on-orbit servicing. This reduces the cost of satellites, and increases flexibility and reliability of the entire system. The other popular myth is that "there is nothing to service", based on the assumption that when a satellite experiences a malfunction, it cannot be repaired to recover operability. This consideration comes primarily from the current practice in satellite design, reliability, and risk management: as I wrote in chapter 2, in space system engineering the failure of components is usually considered absolute, without models for partial failures and partial operability loss, even when fault trees are used and the importance of components on the operability is quantified [59]. Some authors even specified that their analysis of performability, even if applied at the level of components and taking into account the interdependencies, was only applied to non-repairable systems [126,127]. A similar approach is followed by Castet and Saleh, in their statistical data analysis of satellite reliability [96]: this study addresses the criticality of components in terms of their impact on failure, but the failures are always absolute, and the interdependencies between components are not analyzed. Other authors focused on flexibility and and redundancy. Greenberg [95] added maintenance strategy to the simple reliability analysis: satellites have spare redundant parts onboard that can be activated if needed, to maintain the desired level of operability. Also Tafazoli [128] studied on-orbit failures, and the impact of

components on failures. In his recommendations, in addition to the usual testing and redundancy, he added flexibility as a factor to decrease the impact of failures.

Sullivan [125, 129] followed a different approach from the current practice of just analyzing and increasing reliability: he studied failures of United States satellites launched between 1984 and 2003, classified them in categories (launch failures, deployment failures, component failures, unknown failures), and determined the technical and economical feasibility of performing servicing of dysfunctional satellites. He also underlined that since a spacecraft is a complex coupled system, even a single malfunction can trigger cascading failures which may make the system non-operational. Based on the concept of cascading failures, complexity of the system, and impact on cost and safety, the application described in this section considers modular operational satellites. Each module, performing a different function, is connected to the others only through simple interfaces, so that it can be rapidly and easily accessed and/or replaced. Besides the operational satellites, the scenario includes servicing satellites, i.e. a set of satellites capable to perform inspection and maintenance. The servicing satellites, equipped with replacement modules, can perform multiple on-demand servicing operations, thus being the source of flexibility and resilience in this architecture.

The whole architecture is characterized as a two-level hierarchical SoS (figure 4.20): the higher level consists of the servicing satellites, the constellations of operational satellites, and their capabilities. Operational satellites can be satellites for communication, for observation, or for experiments. Each constellation is formed by satellites of the same type. The lower level describes a simple model of the inner architecture of the modular satellites, i.e. the satellite onboard systems and their operational dependencies. These satellites have their own capability, which may be used as part of a constellation. The goal of this application is to analyze the performance, robustness, and resilience of the whole SoS, with different architectures characterized by a different number of servicing satellites in various orbits.

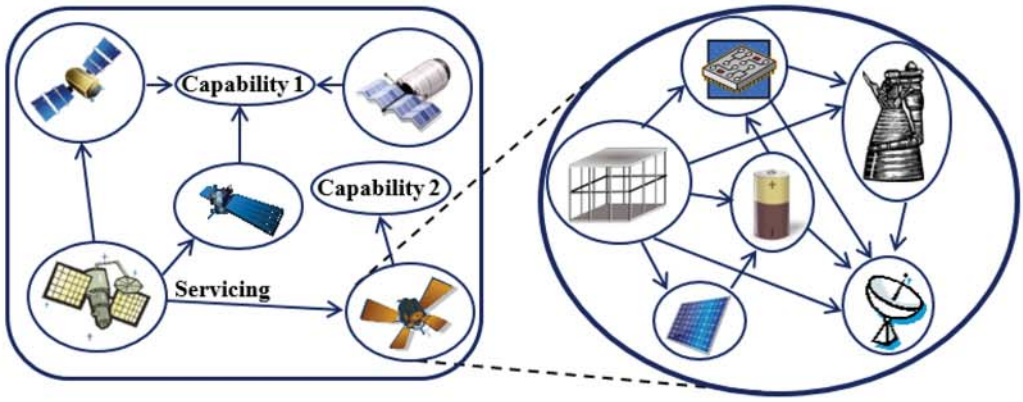


Figure 4.20. On-orbit satellite servicing System-of-Systems. Left: higher level, consisting of operational satellite constellations, capabilities, and servicing satellites. Right: lower level, consisting of satellite modules with their operational dependencies.

At the lower level, the operational dependencies between the component modules of a single satellite are accounted for. Degradation of the modules over time, as well as major disruptions, impact the satellite capability based on the topology and the parameters of a SODA model of the satellite. Different architectures and patterns can therefore be compared based on the robustness of individual satellites to different disruptions. At the upper level, capabilities of individual satellites are combined in other operational networks, representing the constellations, each having its robustness to disruption. If servicing satellites are available, when the operability of a constellation decreases below a given threshold maintenance can be requested. The weakest satellite in the constellation puts out a request for replacement of its weakest module. If a servicing satellite has a replacement module available, and it can maneuver to reach the disrupted satellite with a total impulse Δv lower than a given constraint, the servicing satellite will move to the orbit of the disrupted satellite, and the replaced module will not be available anymore, though the servicing satellite can perform more maintenance operations with other available replacement modules. The architecture which includes servicing satellites exhibits flexibility, and consequently resilience to disruptions.

Lower level

The satellite model is simplified to an operational network of modules, which are supposed to be easily accessible to be replaced or maintained. Simplifying assumptions about the level of detail can be relaxed, and more detailed and realistic models can be used in order to refine the results. Each satellite can belong to one of three types: Communication Satellites, Observation Satellites, and Experimental Satellites. Some of the modules are present in all the satellites, other modules are specific to a certain type of satellite. For some of the systems, alternate choices are available. The orbits available are polar (inclination between 80° and 100°) Low Earth Orbit (LEO)[1], polar and equatorial (inclination between 0° and 5° and between 175° and 180°) Medium Earth Orbit (MEO)[2], Geostationary Earth Orbit (GEO)[3], Molniya[4] and Tundra[5]. Table 4.4 shows the main properties of the satellite types modeled in this study, as well as type and number of modules that constitute the satellite.

Table 4.4.: Properties and number of modules of the operational satellites in the On-orbit satellite servicing System-of-Systems

	Communication satellites	**Observation satellites**	**Experimental satellites**
Orbit	GEO, Molniya, Tundra	LEO, MEO	LEO, MEO, GEO
Structure	1	1	1
Power Controller	1	1	1

continued on next page

[1]Semimajor axis between 6578 and 7378 km. (height between 200 and 1000 km)
[2]Semimajor axis between 8378 and 11378 km. (height between 2000 and 5000 km)
[3]Circular, equatorial, geosynchronous (radius of 42164.140 km)
[4]Semi-geosynchronous, inclination of 63.4°, semimajor axis of 26561.744 km
[5]Inclination of 63.4°, geosynchronous (radius of 42164.140 km)

Table 4.4.: *continued*

	Communication satellites	**Observation satellites**	**Experimental satellites**
Power Source and Storage	1-2 fuel cells without recharge OR solar panels on the external surface with 1-3 batteries OR 1-2 solar arrays with 1-3 batteries		
Power Regulators	2-3	2-3	2-3
Main software	1	1	1
Transponder and gyros	1	1	1
Main software	1	1	1
Guidance, Navigation, and Control (GNC)	1	1	1
Thruster groups	4-6-8	4-6-8	4-6-8
Useful Payload	2-6 communication antennas	3-5 sensors	1-3 experiments

Satellites with different characteristics can be specified by the user, or a random generator can model satellites, with a given probability for the choice of the type, and of the component modules. In this study, I used the random generator to create 25 constellations per satellite type, for a total of 250 satellites (68 communication, 87 observation, and 95 experimental). The random generator can be easily modified according to the user's needs. According to SODA input/output model, the modules

have operational dependencies, and their behavior depends both on their internal status and on the input received. For example, the operability of a battery will depend both on the effectiveness of the battery itself and on the correct operability of the solar arrays. The operability of an observation sensor will be critically dependent on the GNC system, while the criticality of the dependency on one of three onboard regulators will be low. Since the goal of this application is to show possible uses of SODA, rather than to solve this specific problem with high realism, I decided to assign the dependency parameter based on my expertise. Historical surveys of satellite failures and their consequences are available in literature for a more accurate representation of this specific problem.

Figure 4.21 shows a communication satellite created by the random generator: it is in Geostationary orbit, gets power from two solar arrays, and has three batteries, two power regulators, eight thruster groups and four antennas. The figure also gives an idea of the complexity of the operational interdependency between the modules.

For what concerns the self-effectiveness of the systems, I implemented a simple model of the evolution of this value for each module has been implemented. I considered 200 time steps of one month each. Starting from the maximum level of operability, i.e. 100, experiences a decrease in self-effectiveness due to three factors:

1. Aging / wearing out: it is modeled as a small, but always greater than zero, random loss in self-effectiveness. Modules exposed to the space environment (structure and solar arrays) have the highest rate of aging, computing modules have the lowest rate of aging.

2. Degradation due to minor failures: it is modeled with a Beta(1,11) distribution, resulting in possible small losses.

3. Major failures / accidents / catastrophic events: these occur with a very low probability, but produce high loss in self-effectiveness, sampled from an exponential distribution and ranging between 70 and 100.

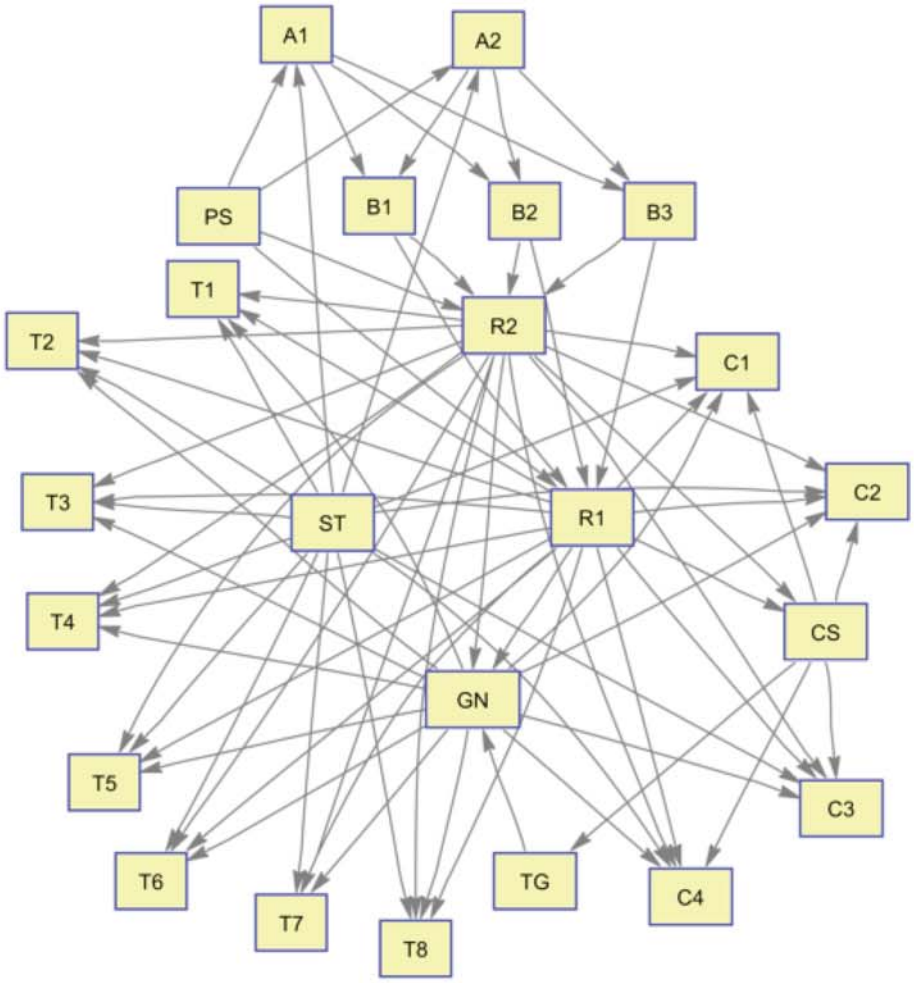


Figure 4.21. Modules of a communication satellites and their operational dependencies. The satellite has structure (ST), power controller (PS), solar arrays (A1 and A2), three batteries (B1-B3), two regulators (R1 and R2), communication software (CS), transponder and gyros (TG), GNC system (GN), eight thruster groups (T1-T8), and four communication antennas (C1-C4).

Figure 4.22 shows one instance of the evolution over time of the self-effectiveness of the modules of the same satellite modeled in figure 4.21. The satellite has twenty-four modules. The bold lines in figure 4.22 highlight some of the concepts used in the evolution model. The communication software keeps its internal status always at a very high level. The solar array, which is subject to weathering from the harsh space

environment, shows a steeper decrease in operability, and some "jump" due to minor disruptions, for example meteorite hits. Battery No. 3 experienced a major failure in this instance, and could be a candidate for on-orbit maintenance.

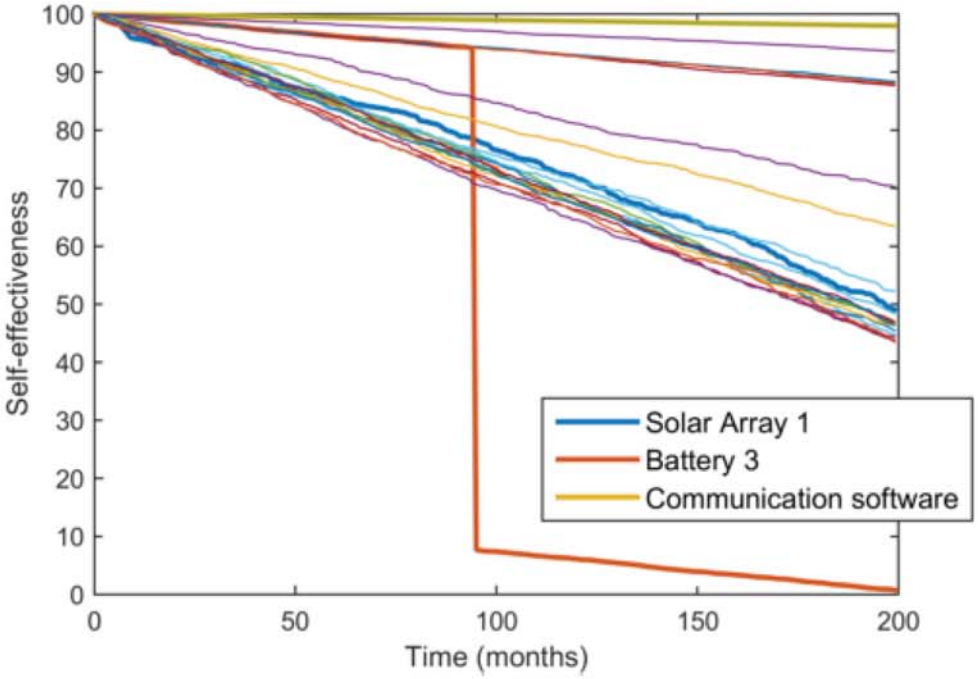


Figure 4.22. Evolution of self-effectiveness of the modules of a communication satellite. Highlighted plots show a module with low aging (communication software), a module with high aging and minor disruptions (solar array), and a module experience a major disruption (battery No. 3)

The evolution of the self-effectiveness over time is used as input to the SODA tool, to quantify the operability of each module over time. The operability of the four communication antennas is the operability of interest in this scenario. Figure 4.23 shows one instance of analysis of the operability of the communication satellite. Two modules, battery No. 2 and thruster group No. 1, experience a major disruption in this instance. The other modules exhibit only aging and minor losses, and are similar to the plots in figure 4.22. The operability is the average operability of the four communication antennas. The partial drop in the operability of the battery affects

the operability of the whole satellite, but since the satellite has two more batteries, the impact is limited. The loss of one of eight thruster groups is almost entirely absorbed by the satellite, thanks to the dependency between the modules.

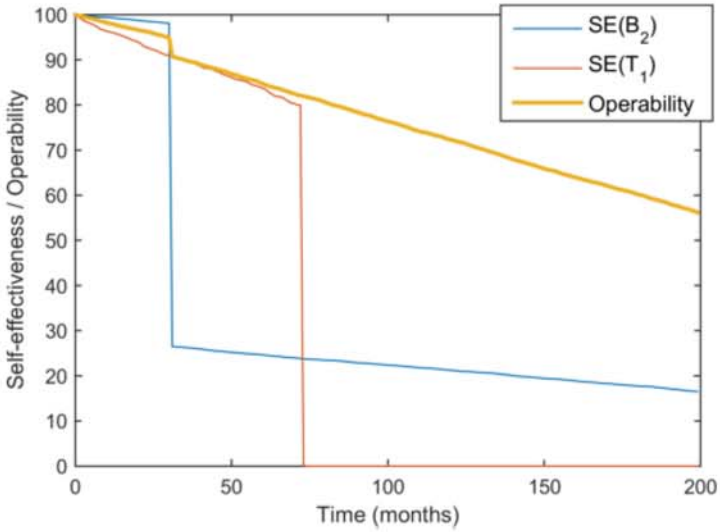


Figure 4.23. Evolution of operability of one instance a communication satellite with two modules experiencing major failures.

Due to the probabilistic nature of the evolution of the modules, I analyzed 250 instances of the life cycle of this communication satellite. Figure 4.24 shows the operability over time in the best case (instance with only a minor failure), the average operability of the 250 instances, and the operability over time in the worst case (instance where a major structural failure occurred soon in the satellite life). Table 4.5 shows some of the results of SODA analysis. This satellite has a robustness of 0.605, and after 100 months only 32.4% of the instances show an overall operability lower than 70. As expected, the most exposed systems and modules were the critical cause of the low operability, with the two solar arrays and the satellite structure being the most critical part in almost half of the instances.

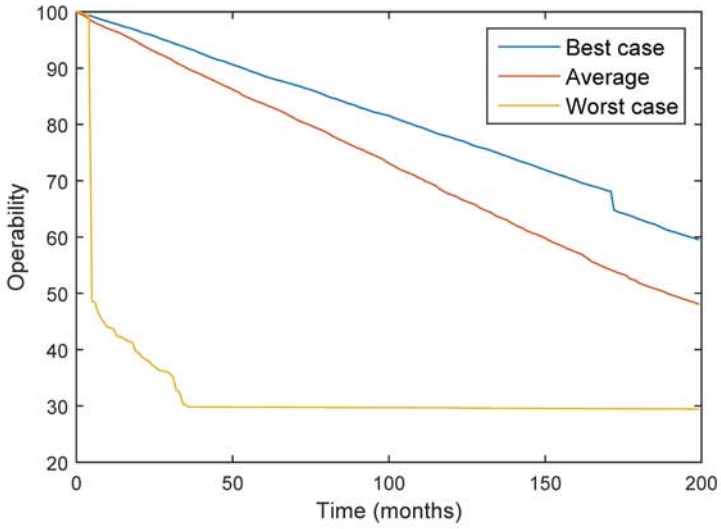


Figure 4.24. Operability of a communication satellite in the best and worst case and average operability over time.

Table 4.5. Results of SODA analysis of a single communication satellite.

Percentage of instance below operability threshold of 80 after 100 months	75.2%
Percentage of instance below operability threshold of 70 after 100 months	32.4%
Percentage of instance below operability threshold of 70 after 200 months	100%
Most critical systems	Solar array No. 2 (16.0% of cases) Structure (14.8% of cases) Solar array No. 1 (14.4% of cases) Transponder and gyros (7.4% of cases) Thrusters group No. 1 (7.3% of cases)
Robustness	0.605

Higher level

As mentioned before, at the higher level for this application I generated 25 constellations per satellite type, for a total of 250 satellites (68 communication, 87 observation, and 95 experimental). The number of modules and systems per satellite ranges from 14 to 27, for a total of 5123 subsystems in the whole SoS. Results from the lower level are combined in the higher level, where the operability of the payload modules of the satellites in a constellation contribute to the overall operability of the constellation. At the higher level, I also considered on-orbit servicing. This allows for evaluation and computation not only of the robustness of the constellations, but also of the resilience, and of the gain in operability achieved thanks to servicing, vs. the impulse required to perform the transfer maneuvers to service the operational satellites. The servicing satellites start in LEO, MEO, or GEO orbits, and have a different number and type of spare modules. Some components of the operational satellite, for example the structure in case of large failure, is considered not serviceable.

In this SODA analysis, I made the following simplifying assumptions:

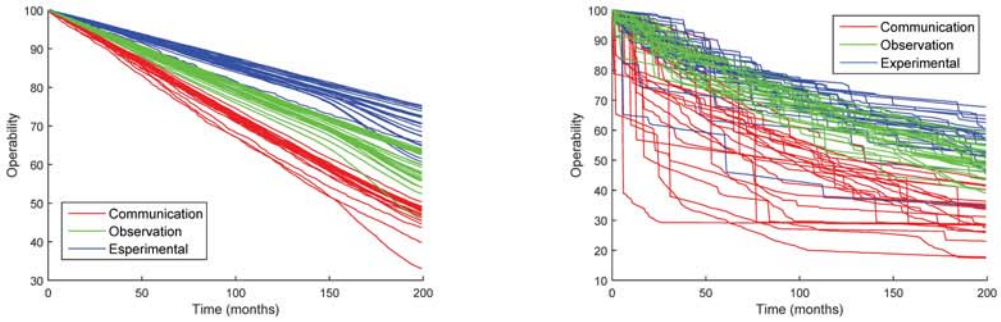
- Servicing satellites are not subject to failures, so they are not decomposed into subsystems.
- The cost analysis accounts only for the impulse Δv required to maneuver between the current orbit of the servicing satellite and the orbit of the target operational satellite. More complex fuel and cost analysis can be found in [125], and studies for optimal rendezvous transfers can be found for example in [130].
- SODA analysis is used only for analysis and comparison of architectures, and not to guide the design of operational satellites.
- The stakeholders of servicing satellites offer their services whenever it is possible without violating the rule of the maximum Δv.
- The servicing is always effective, and the weakest module in the appropriate satellite is replaced with a new one, having maximum self-effectiveness.

Depending on the satellite stakeholders, these assumptions can be modified, thus properly addressing gap 1 from section 2.5. Table 4.6 lists the number of constellations per type per number of satellites.

Table 4.6. Number of constellations per type per number of component satellites.

	Number of satellites						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Communication	7	6	5	3	2	2	0
Experimental	1	4	5	8	4	2	1
Observation	2	2	8	9	3	1	0

First of all, I ran 250 instances of a scenario with all 75 constellations, without considering the servicing satellites. Similarly to what done for a single satellite, the evolution of the operability of each constellation over time is computed on each instance, and yields information about the robustness of the constellation. Figure 4.25 shows the mean operability over time of the 75 constellations, and the operability over time in the worst case for each of the 75 constellations. The average operability (figure 4.25(a)) shows a clear pattern driven by the type of satellites in the constellation. Constellations of communication satellites in this model have a lower number of satellites, and the satellites themselves have a low number of payload modules. For this reason, failures in communication satellites have a larger impact than failures in other types of satellites. At the other end of the spectrum, constellation of experimental satellites consist a larger number of satellites, which increases the robustness of the constellation. The only constellation having an average operability similar to that of constellations of observation satellites consists of a single satellite. Operability in the worst case for each constellation (figure 4.25(b)) exhibits a similar pattern, with major malfunctions in the satellites clearly showing a substantial impact on the operability of the constellation.



(a) Average operability of the constellations over time

(b) Operability in the worst case for each constellation over time

Figure 4.25. Average operability and operability in the worst case for each satellite constellation. Constellations are marked according to the type of the component satellites.

I used results from the runs and analysis to fill a table listing the robustness, average operability and operability in the worst case for each constellation, without servicing. Table 4.7 confirms again that in general high robustness is associated with constellations having a higher number of satellites. However, the results yield even more information. Communication constellations exhibit a more gradual increase in robustness with increasing number of satellites. Experimental and observation constellation show a relatively large increase in robustness when the number of satellites increases from 1 to 2, then is stays almost constant for increasing number of satellites. There are also a few exception, i.e. constellations that have robustness very dissimilar to that of other constellations having the same number of satellites. These cases can be studied in more detail by examining the self-effectiveness and interaction of their modules of the lower level. This detailed analysis is beyond the scope of this application within this dissertation, therefore it is not reported here.

The expected value of the average operability over time and the average operability over time in the worst case is also listed in the table. While these two values follow in general the pattern of the robustness, it is interesting to notice how the use of absolute operability, unrelated to the amount of disruption, can be deceiving. For

example, constellation 17 has a higher level of average operability with respect to constellation 13. However, the robustness shows that constellation 13 can react better to disruptions, suggesting that the high level of average operability of constellation 17 is due to a level of losses in self-effectiveness that is in average lower to the losses experienced by constellation 13.

Table 4.7.: Properties of the constellations without servicing. Constellations 1-25: communication satellites. Constellations 26-50: experimental satellites. Constellations 51-75: observation satellites.

Const.	**No. of sats**	*Rob*	**E(O)**	**min(O)**	**Const.**	**No. of sats**	*Rob*	**E(O)**	**min(O)**
1	2	0.68	74.7	49.5	39	4	0.85	85.7	74.8
2	1	0.57	67.1	30.1	40	7	0.85	86.0	76.4
3	3	0.70	73.8	58.5	41	3	0.86	87.6	74.2
4	2	0.67	73.3	54.3	42	4	0.86	87.0	78.0
5	2	0.66	73.4	52.3	43	3	0.83	84.8	68.6
6	6	0.71	73.4	56.4	44	4	0.86	87.3	77.4
7	2	0.68	72.5	50.3	45	4	0.82	83.0	72.8
8	1	0.55	69.8	38.0	46	6	0.84	84.8	77.1
9	3	0.68	72.8	56.2	47	4	0.85	85.7	74.2
10	4	0.70	72.2	56.0	48	5	0.86	86.7	74.4
11	3	0.69	72.1	55.1	49	4	0.84	85.0	72.8
12	5	0.71	73.6	61.8	50	2	0.84	86.9	71.8
13	5	0.71	73.1	55.9	51	3	0.77	79.1	70.8
14	1	0.58	71.8	43.3	52	5	0.79	80.1	72.3

continued on next page

Table 4.7.: *continued*

Const.	**No. of sats**	*Rob*	**E(O)**	**min(O)**	**Const.**	**No. of sats**	*Rob*	**E(O)**	**min(O)**
15	3	0.68	72.3	55.8	53	4	0.79	81.4	74.5
16	1	0.59	71.8	31.6	54	1	0.74	81.1	65.3
17	1	0.61	73.2	46.4	55	6	0.79	80.8	75.2
18	1	0.60	73.1	48.8	56	4	0.77	78.5	65.1
19	6	0.71	72.7	59.1	57	5	0.78	79.5	72.1
20	4	0.71	73.4	61.4	58	3	0.75	76.8	65.1
21	2	0.66	73.3	52.9	59	4	0.78	79.4	69.3
22	3	0.69	73.2	58.0	60	3	0.77	79.2	69.3
23	4	0.70	73.2	60.5	61	2	0.75	78.5	67.1
24	1	0.59	68.4	35.5	62	3	0.77	79.1	71.6
25	2	0.66	73.7	53.6	63	4	0.78	79.4	71.5
26	2	0.81	83.5	62.7	64	4	0.78	79.4	68.1
27	3	0.83	84.2	70.4	65	3	0.73	75.4	64.7
28	3	0.86	87.5	73.1	66	4	0.79	81.6	71.7
29	5	0.84	85.2	77.5	67	4	0.79	81.4	74.0
30	2	0.85	86.9	71.0	68	4	0.76	77.8	66.5
31	6	0.84	85.2	76.3	69	2	0.76	80.7	67.9
32	4	0.85	86.1	76.0	70	3	0.78	81.1	72.8
33	3	0.84	85.1	73.4	71	3	0.79	81.4	72.6
34	4	0.85	85.9	71.5	72	4	0.78	79.2	70.7
35	5	0.84	85.2	73.6	73	3	0.76	78.9	69.5
36	1	0.77	85.2	47.1	74	1	0.72	81.3	64.3

continued on next page

Table 4.7.: *continued*

Const.	**No. of sats**	*Rob*	**E(O)**	**min(O)**	**Const.**	**No. of sats**	*Rob*	**E(O)**	**min(O)**
37	5	0.85	85.9	77.4	75	5	0.79	81.1	74.2
38	2	0.82	84.2	68.0					

The introduction of servicing satellites brings resilience into the SoS, that can sometime react to disruptions and failures and recover operability, at least partially. The first configuration I tested has 30 servicing satellites, 10 in GEO, 10 in MEO, 10 in LEO. The threshold of the constellation operability that triggers a request for operability is 80. The maximum Δv allowed for a single transfer towards an operational satellite requesting servicing is $1000\,m/s$. This value, together with the spare parts available on board of servicing satellites, influences the percentage of requests satisfied.

Figure 4.26 shows an examples of the resulting operability in one instance of two constellations. Some satellite in constellation 8 experiences a major malfunction that triggered a request, but an appropriate servicing satellite was not available. Instead, constellation 21 exhibits major loss in operability at multiple time steps. The first of these losses that brings the operability below the threshold triggers a request that is answered, therefore the constellation recovers operability above the threshold. Later losses, however, are not recovered. This can be caused either by lack of appropriate spare modules in the original configuration of the servicing satellites or by unavailability of spare modules because used for maintenance on another operational satellite.

Figure 4.27 shows the average operability of all the constellations, grouped by satellite type, without servicing satellites and with servicing satellites. Communication and observation constellations exhibit the highest increase in operability due to servicing, while experimental constellations have only a marginal gain.

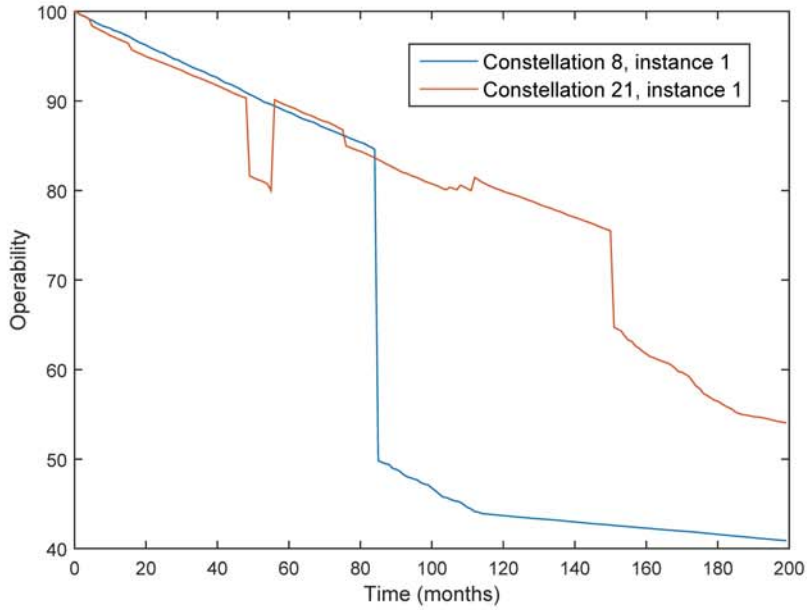
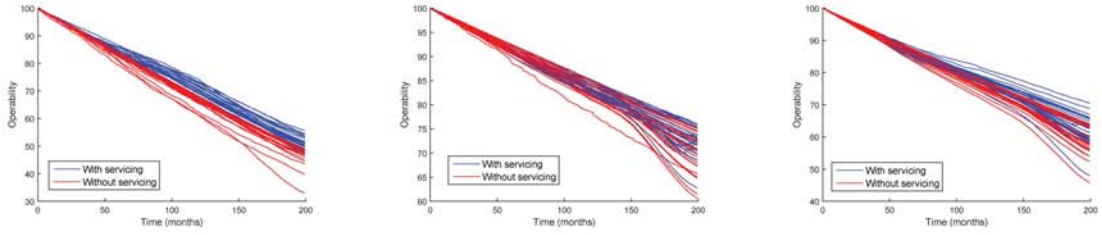


Figure 4.26. Operability of one instance of constellations 8 and 21 with servicing available.



(a) Average operability of communication constellations over time, without and with servicing

(b) Average operability of experimental constellations over time, without and with servicing

(c) Average operability of observation constellations over time, without and with servicing

Figure 4.27. Average operability of the constellations with and without servicing.

Table 4.8 lists the average number of requests per satellite, the average resilience, the average gain in operability and the gain in operability in the worst case for each constellation. The number of requests per satellite is lower in the constellations with a larger number of satellite, due to their higher robustness. Higher level of resilience

occur in constellations with a low number of satellites. Sometimes the average gain in operability is little, because not all requests can be satisfied. The gain in the worst case is in general higher than the average gain, but for communication constellations the worst case often does not receive servicing.

Table 4.8.: Properties of the constellations with servicing. Constellations 1-25: communication satellites. Constellations 26-50: experimental satellites. Constellations 51-75: observation satellites.

Const.	**No. Reqs per sat**	*Res*	**gain in E(O)**	**gain in min(O)**	**Const.**	**No. Reqs per sat**	*Res*	**gain in E(O)**	**gain in min(O)**
1	2.7	0.70	2.6	4.7	39	0.7	0.85	0.1	1.6
2	1.4	0.57	0	0	40	7	0.4	0.1	0
3	2.2	0.72	2.1	2.6	41	1.3	0.86	0.2	2.2
4	1.0	0.70	2.5	0	42	0.7	0.86	0.1	0
5	1.1	0.68	1.8	0	43	1.1	0.84	0.3	0
6	0.8	0.72	1.1	0	44	0.3	0.86	0	0.3
7	2.0	0.70	1.8	0	45	1.2	0.82	0.7	0.5
8	2.3	0.55	0	0	46	1.2	0.84	0.2	1.2
9	0.7	0.71	2.2	3.4	47	1.3	0.85	0.3	0
10	1.0	0.70	0.8	0.6	48	0.6	0.86	0.1	0
11	1.5	0.72	2.7	0	49	1.2	0.84	0.4	1.0
12	1.6	0.72	1.5	0	50	0.5	0.85	0.4	0
13	0.7	0.73	1.9	0	51	1.0	0.77	0.6	0
14	0.9	0.58	0	0	52	1.3	0.79	0.6	0.9

continued on next page

Table 4.8.: *continued*

Const.	**No. Reqs per sat**	*Res*	**gain in E(O)**	**gain in min(O)**	**Const.**	**No. Reqs per sat**	*Res*	**gain in E(O)**	**gain in min(O)**
15	2.1	0.71	2.3	0	53	0.7	0.80	0.8	1.3
16	4.9	0.64	5.5	0	54	0.8	0.76	2.4	2.7
17	0.9	0.66	4.2	2.0	55	1.1	0.80	0.3	0.2
18	1.0	0.60	0	0	56	4	1.0	1.0	7.1
19	1.3	0.72	0.8	0	57	1.6	0.79	0.9	0.3
20	1.0	0.72	1.1	0	58	1.0	0.77	2.0	3.2
21	1.2	0.69	3.2	0	59	0.7	0.79	1.1	4.8
22	1.5	0.70	1.2	0	60	0.8	0.78	1.4	2.1
23	2.3	0.71	1.4	0.4	61	1.6	0.78	2.7	1.9
24	0.7	0.67	7.2	5.8	62	2.5	0.78	1.5	0
25	1.6	0.69	2.9	3.8	63	0.8	0.79	1.5	2.5
26	0.5	0.83	2.0	3.6	64	1.7	0.79	1.0	3.6
27	0.3	0.84	1.5	2.4	65	1.5	0.74	1.3	0.6
28	0.6	0.86	0.2	0	66	1.1	0.80	0.8	2.0
29	1.7	0.84	0.1	0.9	67	1.7	0.80	0.8	0.4
30	0.5	0.85	0.5	0	68	1.0	0.78	2.2	5.6
31	0.5	0.85	0.5	0	69	1.0	0.78	1.8	5.5
32	2.6	0.86	0.2	0.5	70	1.9	0.79	0.8	0
33	0.4	0.84	0.4	1.2	71	0.9	0.79	0.8	0.3
34	0.6	0.85	0.1	1.4	72	0.8	0.78	0.7	0
35	1.3	0.85	0.2	0.3	73	1.0	0.78	1.0	0.5

continued on next page

Table 4.8.: *continued*

Const.	**No. Reqs per sat**	*Res*	**gain in E(O)**	**gain in min(O)**	**Const.**	**No. Reqs per sat**	*Res*	**gain in E(O)**	**gain in min(O)**
36	0.6	0.81	3.5	9.8	74	0.6	0.76	3.0	2.8
37	0.9	0.86	0.5	1.0	75	1.0	0.80	0.6	1.1
38	3.4	0.83	1.4	4.8					

Table 4.9 shows the results of the analysis of various architectures for on-orbit servicing. The architecture have a different number of servicing satellites. This analysis leads the way to more detailed evaluations, and can guide decision-making in design and architecture of on-orbit servicing SoS. For each architecture, 250 instances of 200 timesteps have been run.

First of all, results show that the architecture with 10 servicing satellites per orbit type satisfies 32% of the requests, while the architecture with 5 servicing satellites per orbit type satisfies 23.4%. Since the average Δv per maneuver is about the same, the cheaper architecture with less servicing satellites can be preferable, given that the increase in operability is very similar, especially in the worst case. One possible improvement is to use SODA to calculate offline the expected improvement given by the servicing, and performing the requested maintenance only if the gain is above a given threshold.

Architectures with servicing satellites only on one type of orbit also exhibit interesting results. Servicing in GEO results more effective, and it is cheaper because all the satellites are on the same orbit, so that only phasing maneuvers are requires. Satellites in LEO and in MEO are spread over a wide range of orbit inclination, therefore requiring expensive maneuvers to change orbital plane. Besides modifying the

Δv threshold, the user can use satellite catalogues to decide optimal orbits for the servicing satellites.

Table 4.9. Number of constellations per type per number of component satellites.

Servicing architecture	**Avg no. reqs (200 months)**	**Avg % satisfied requests**	**Avg Δv per maneuver** (m/s)	**Avg increase in E(O)**	**Avg increase in min(O)**
30 servicing satellites, 10 GEO, 10 MEO, 10 LEO	283.5	32.0	307.1	1.3	17.6
15 servicing satellites, 5 GEO, 5 MEO, 5 LEO	242.9	23.4	306.2	0.9	16.5
15 servicing satellites in GEO	214.9	16.4	56.2	0.8	20.6
15 servicing satellites in MEO	219.3	17.1	560.5	0.45	8.6
15 servicing satellites in LEO	215.3	18.2	555.5	0.53	11.1

4.2 Developmental analysis of simple space architectures

"There cannot be a crisis next week. My schedule is already full."

Henry Kissinger, quoted in The New York Times Magazine
1 June 1969

4.2.1 Communication satellite

In this section, I illustrate a simple use of SDDA to quantify high-level systems criticalities and delay absorption in the development of a communication satellite, analogous to the satellite whose operational dependencies are shown in figure 4.21, but with redundant systems grouped in a single node. The developmental dependencies of the systems on board of the satellite do not generally coincide with the operational dependencies. Figure 4.28 shows the component systems and their developmental dependencies for this application, which addresses gaps 2 and 4 from section 2.5.

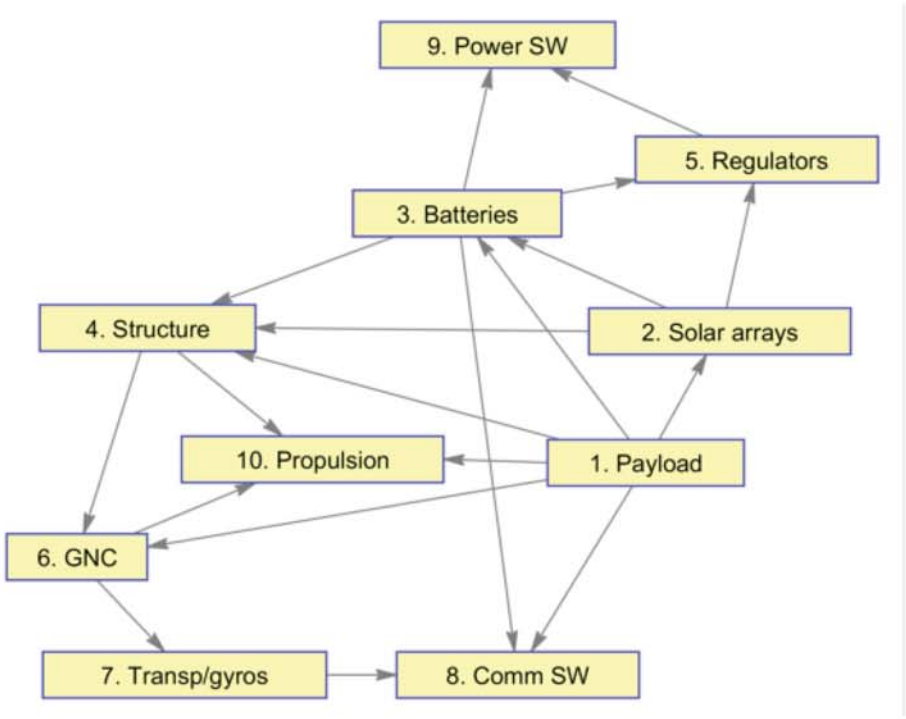


Figure 4.28. Developmental dependencies of systems onboard a communication satellite.

As described in section 3.2, developmental dependencies in SDDA are similar to dependencies in a PERT network, but SDDA models not-absolute dependencies, allowing for partial overlapping in the development schedule. Since a system can begin its development with a *lead time*, the whole development process can absorb part of delays that might occur in the development of some of the systems involved. The values of strength and criticality of the developmental dependencies in the communication satellite are as follows (the node numbers represent the modules in figure 4.28):

$$SOD = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \\ \mathbf{8} \\ \mathbf{9} \\ \mathbf{10} \end{array} \begin{array}{c} \begin{array}{cccccccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} \end{array} \\ \begin{bmatrix} 0 & 0.4 & 0.5 & 0.25 & 0 & 0.15 & 0 & 0.7 & 0 & 0.3 \\ 0 & 0 & 0.5 & 0.3 & 0.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.2 & 0.6 & 0 & 0 & 0.5 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$COD = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \\ \mathbf{8} \\ \mathbf{9} \\ \mathbf{10} \end{array} \begin{array}{c} \begin{array}{cccccccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} \end{array} \\ \begin{bmatrix} 0 & 20 & 40 & 15 & 0 & 30 & 0 & 25 & 0 & 30 \\ 0 & 0 & 50 & 30 & 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 35 & 0 & 0 & 20 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 25 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The minimum and maximum development time of the systems are (in weeks)

$$t_{min} = \begin{bmatrix} 14 \\ 12.5 \\ 6 \\ 11 \\ 3 \\ 2.5 \\ 15 \\ 4 \\ 3 \\ 9 \end{bmatrix} \qquad t_{max} = \begin{bmatrix} 24 \\ 17 \\ 12 \\ 23 \\ 5 \\ 4 \\ 21 \\ 5 \\ 4 \\ 12 \end{bmatrix}$$

Figure 4.29 shows the effect of partial developmental dependency. While PERT considers all the dependencies to be absolute, meaning that each node must wait until all its predecessors are fully developed, SDDA and SDDAmax consider partial overlapping based on the parameters that model the dependencies.

Figure 4.29 shows how the development can be completed in 37.5 weeks, when no delays arise, while PERT computation resulted in 65 weeks expected for development time, due to the absolute dependency between systems. However, the figure also shows how early development causes a few systems, including the GNC (node 6), the propulsion system (node 10) and the communication software (node 8), to have to wait for the completion of other systems, extending their own development over a longer span. While this can result in early achievement of partial capabilities and increase the possible absorption of delays, it also causes an increase in cost. Decision makers must trade-off between these competing aspects.

Delay absorption

Besides the baseline schedule, SDDA analysis can also be used to quantify the delay absorption throughout the development of a complex system, and to identify the most critical elements in terms of impact on the development time. When the

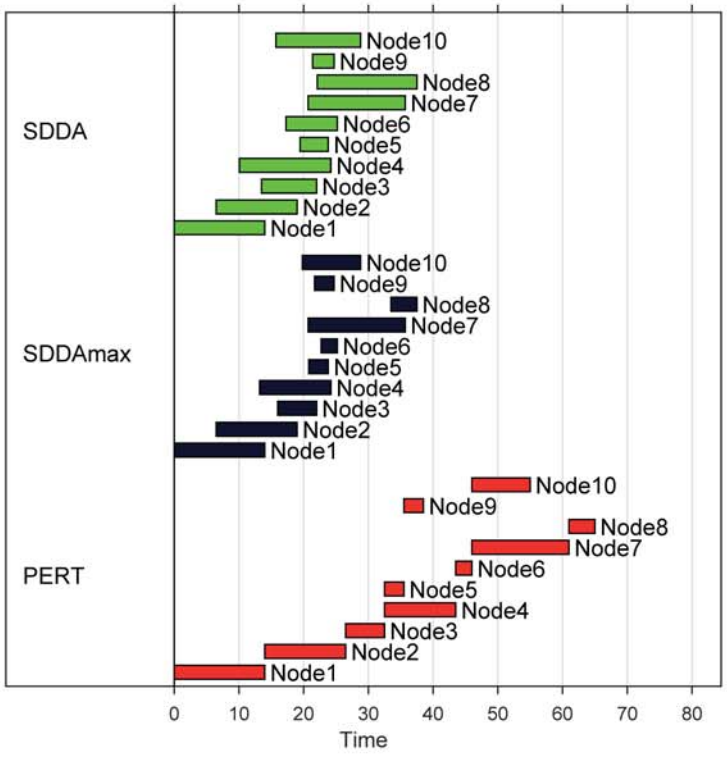


Figure 4.29. Gantt chart of the development of a communication satellite, comparing SDDA model, SDDAmax model, and PERT.

development of a system slows down, the delay affects the development of other systems in the network. In PERT, the delay in a system reverberates entirely on the development of the dependent systems. Delays on the critical path cannot be recovered. If the dependencies are not absolute, i.e. only part of the development of a system depends on the predecessor, the development process of this system can be partially executed even when it is lacking some of the inputs from the systems from which it depends. This results in both a faster overall development, and in possible partial delay absorption. Table 4.10 lists the results of deterministic SDDA analysis, when one system at a time experiences a delay, with its self-effectiveness down to 30, while the other systems are all keeping their fastest development.

Table 4.10. Delay absorption in the development of the communication satellite.

Delayed node	**SDDA**		**SDDAmax**		**PERT**	
	Final time (weeks)	**Delay recovery (%)**	**Final time (weeks)**	**Delay recovery (%)**	**Final time (weeks)**	**Delay recovery (%)**
Payload	51.1	0	51.1	0	72.0	0
Solar arrays	40.7	0	46.5	0	68.2	0
Batteries	39.6	50	48.5	0	69.2	0
Structure	42.8	37.1	47.4	0	73.4	0
Regulators	37.5	100	37.5	100	65	100
GNC	41.4	0	42.1	0	66.1	0
Transp/gyros	41.7	0	43.3	0	69.2	0
Comm SW	37.8	55	38.2	0	65.7	0
Power SW	37.5	100	37.5	100	65	100
Propulsion	37.5	100	37.5	100	65	100

Results show how the partial overlapping of development schedule in SDDA allows for total or partial delay recovery in nodes that are along the critical path (when PERT does not recover any delay). The more conservative SDDAmax, which has less partial overlapping of development schedule (section 3.2.2), exhibits less delay recovery. However, this recovery is measured as percentage with respect to the delay in the affected node. Even if this particular delay causes less overlapping and it is not absorbed, especially in the case of critical dependencies, the final development time with the SDDAmax model is still shorter than the final development time that the complex system would have due to absolute developmental dependencies. SDDA model indicates that delays in the development of the payload heavily impact the

overall development. Delays in the development of the structure, GNC system and transponder/gyros cause some delay in the final development. Thanks to partial parallel development, however, delays in the development of the propulsion system and of the software can be partially or completely absorbed.

Figure 4.30 shows one application of stochastic analysis, according to the methodology described in 3.2. The development uncertainty of batteries, structure, communication software and power system software is low. The development uncertainty of payload, regulators, trasponder/gyros and propulsion is medium. The development uncertainty of solar arrays and GNC is high. The uncertainty in the completion time of the systems causes uncertainty on the "best" beginning time of other systems. The user can use this information to decide when to begin the development of each system, based on current information and amount of accepted risk. The expected value of completion under this scenario is 39.9 weeks in SDDA, 41.3 weeks in SDDAmax, and 66.9 weeks in PERT.

Figure 4.31 shows the reduction in uncertainty when the analysis is conducted again after 20 weeks. The improved information decreases the uncertainty, and the resulting expected value of completion under this scenario is 38.2 weeks in SDDA, 38.8 weeks in SDDAmax, and 66.2 weeks in PERT.

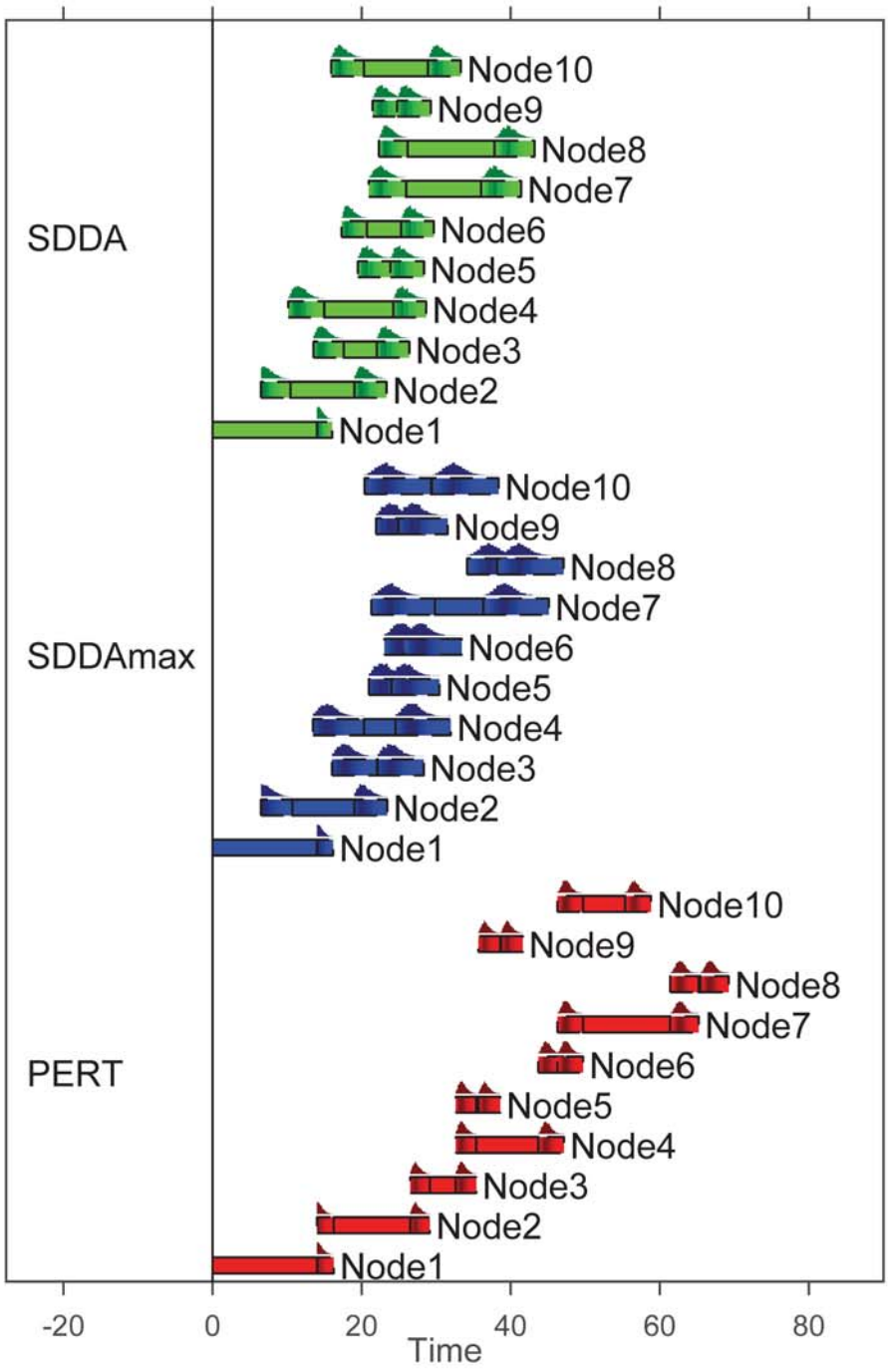


Figure 4.30. Gantt chart for communication satellite under uncertainty. Information at time=0.

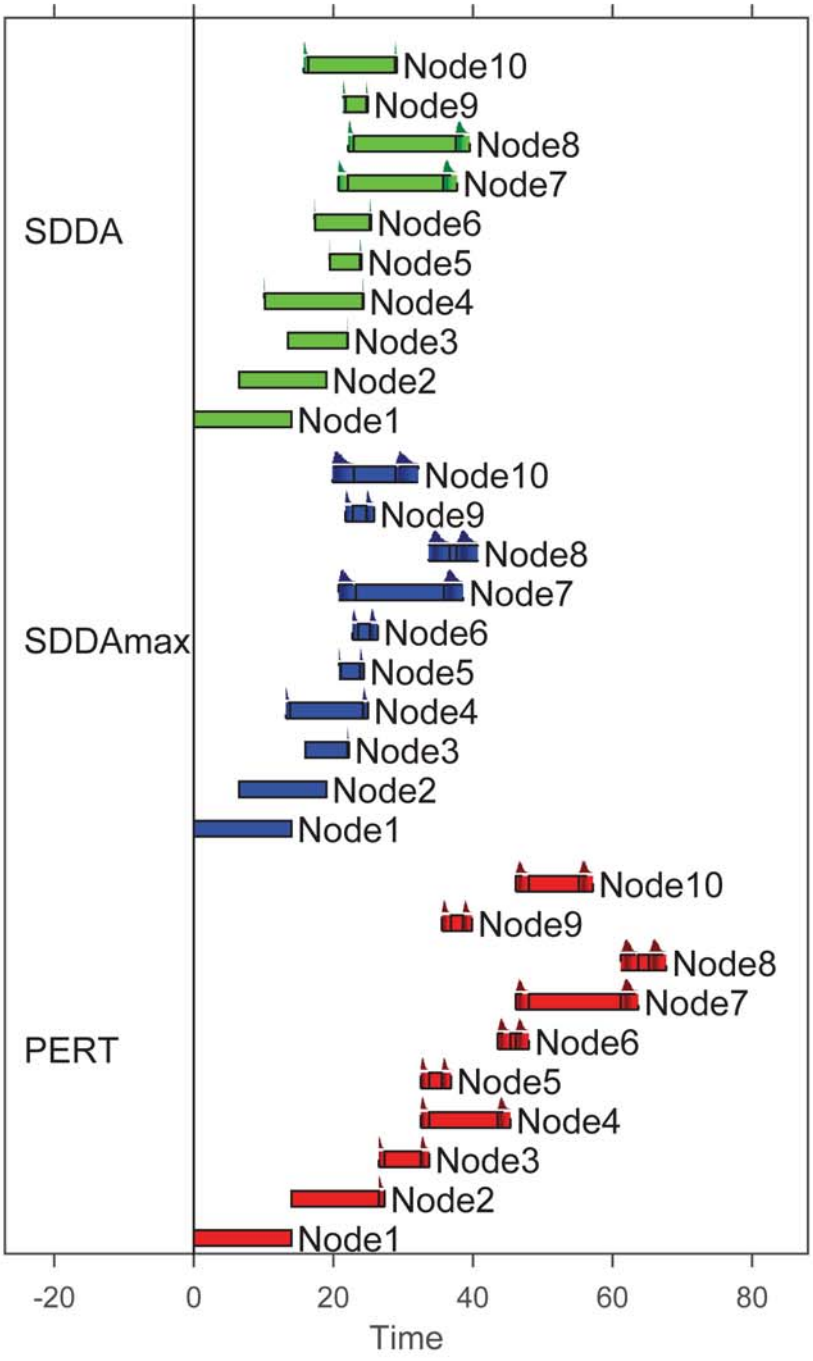


Figure 4.31. Gantt chart for communication satellite under uncertainty. Information at time=20.

4.3 Combined Operational and Developmental Analysis

"A great ship asks deep waters."

George Herbert, *Jacula Prudentum; or Outlandish Proverbs, Sentences, &c, 1651 (§445)*

4.3.1 Naval Warfare Scenario

In this section, I show results of the application of SDDA for development schedule and risk analysis, together with SODA analysis for considerations on partial capabilities. The scenario analyzed with SODA and SDDA is a small Naval Warfare SoS, based on the Littoral Combat Ship (LCS) concept [131]. This SoS is composed of three main systems: a Surface Warfare system, an Anti-Mine system, and an Anti-Submarine system. Each of these systems has ships, helicopters, unmanned vehicles, and other subsystems, listed in table 4.11. The analysis begins with SDDA, whose output will become an input to SODA, to measure operability as more and more systems get developed and deployed. In the same table, I also list the category of each subsystem, and a number that represents the subsystem in the different developmental networks. The objectives of this case study are:

- To perform deterministic and stochastic SDDA analysis.
- To illustrate how to compare different developmental architecture and obtain insights into the development time and risk of delays associated with various architectures and disruptions.
- To illustrate how to use SDDA analysis for developmental policy support.
- To show how to use the results of SDDA as input to SODA, to compare developmental architectures in terms of partial capabilities.

This application addresses all the gaps from section 2.5.

Table 4.11. Systems and Subsystems in the Naval Warfare Scenario SoS.

Node number	**System**	**Category**
1	Ship weaponry	Surface
2	Ship radar	Anti-submarine
3	Ship weaponry	Anti-submarine
4	Ship radar	Anti-mine
5	Ship weaponry	Anti-mine
6	Helicopter radar	Surface
7	Unmanned Air Vehicle radar	Surface
8	Helicopter weaponry	Anti-submarine
9	Remotely Controlled Vehicle radar	Anti-submarine
10	Unmanned Water Vehicle radar	Anti-submarine
11	Helicopter weaponry	Anti-mine
12	Remotely Controlled Vehicle radar	Anti-mine
13	Unmanned Water Vehicle radar	Anti-mine

I analyzed three development architectures, representing different approaches, where various stakeholders participate into the SoS at different times. Development choices, and parameters of dependency are based on availability of resources, and considerations about the efficacy of deployed systems during the development (the user will want to avoid architectures where only radars or only weapons are developed first).

For all these architectures, the minimum and maximum development time of the systems is:

$$t_{min} = \begin{bmatrix} 30 \\ 25 \\ 30 \\ 25 \\ 30 \\ 40 \\ 20 \\ 40 \\ 22 \\ 24 \\ 40 \\ 22 \\ 24 \end{bmatrix} \qquad t_{max} = \begin{bmatrix} 48 \\ 40 \\ 48 \\ 40 \\ 48 \\ 60 \\ 32 \\ 60 \\ 32 \\ 30 \\ 60 \\ 32 \\ 30 \end{bmatrix}$$

Architecture A

In architecture A, all the littoral combat ships, including anti-submarine and anti-mine radars and weaponry in all three categories, are developed first. Then development of the surface systems follows, then the anti-submarine system, and finally the anti-mine system.

The matrices of the parameters modeling the developmental dependencies in architecture A are:

$$
{}^{dev}SOD^{A} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0.6 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.15 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.7 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0.75 & 0.65 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0.4 & 0.4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.55 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.65 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

$$
{}^{dev}COD^{A} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 40 & 40 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 & 35 & 45 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 40 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 55 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 30 & 30 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 35 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Figure 4.32 shows the developmental dependencies between the systems, according to the choices of the developmental manager in architecture A.

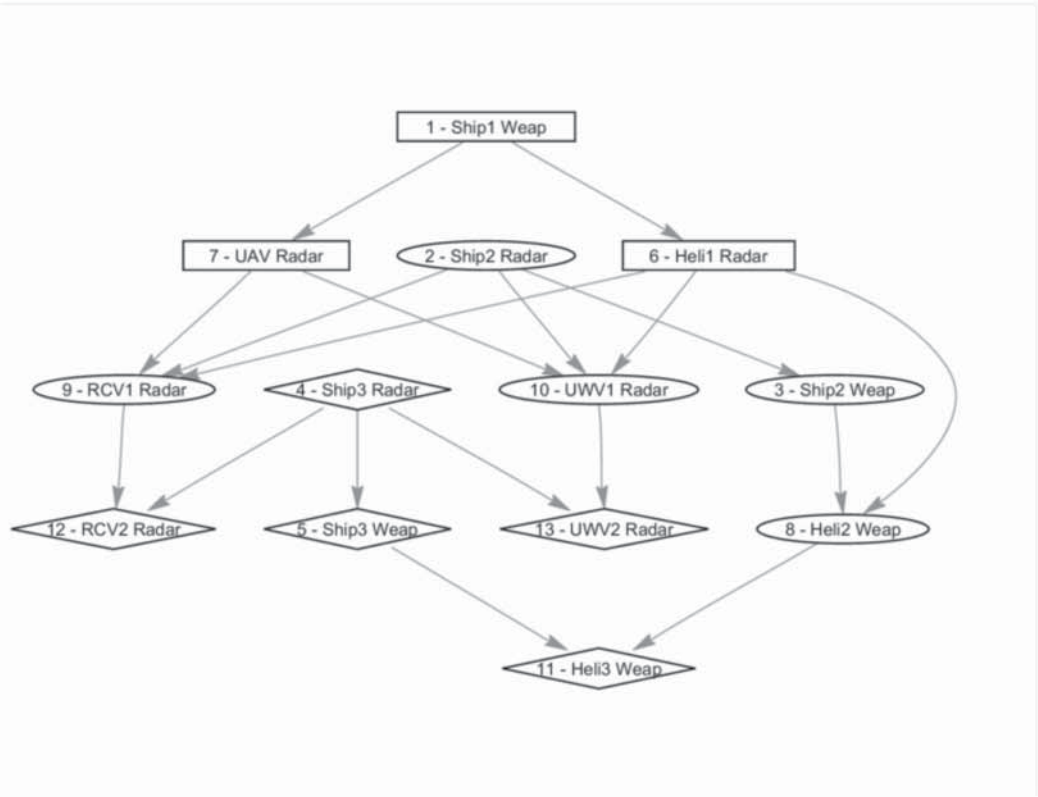


Figure 4.32. Developmental dependencies in architecture A of the Naval Warfare Scenario. Weap = weaponry. UAV = Unmanned Air Vehicle. Heli = MH60 helicopter. RCV = Remotely Controlled Vehicle. UWV = Unmanned Water Vehicle. Rectangles: surface systems. Ellipses: Anti-submarine systems. Diamonds: Anti-mine systems. The ships are developed first.

Architecture B

In architecture B, the surface and anti-mine systems are developed first in their entirety, followed by the anti-submarine system, which requires information input from the first two categories.

The matrices of the parameters modeling the developmental dependencies in architecture B are:

$$
{}^{dev}SOD^{B} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.15 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.7 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0.75 & 0.65 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.6 & 0.55 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.55 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.65 & 0 & 0 & 0
\end{bmatrix}
$$

$$
{}^{dev}COD^{B} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 & 35 & 45 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 40 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 55 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 40 & 65 & 0 & 0 & 0 & 0 & 0 \\
0 & 30 & 35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0
\end{bmatrix}
$$

Figure 4.33 shows the developmental dependencies between the systems, according to the choices of the developmental manager in architecture B.

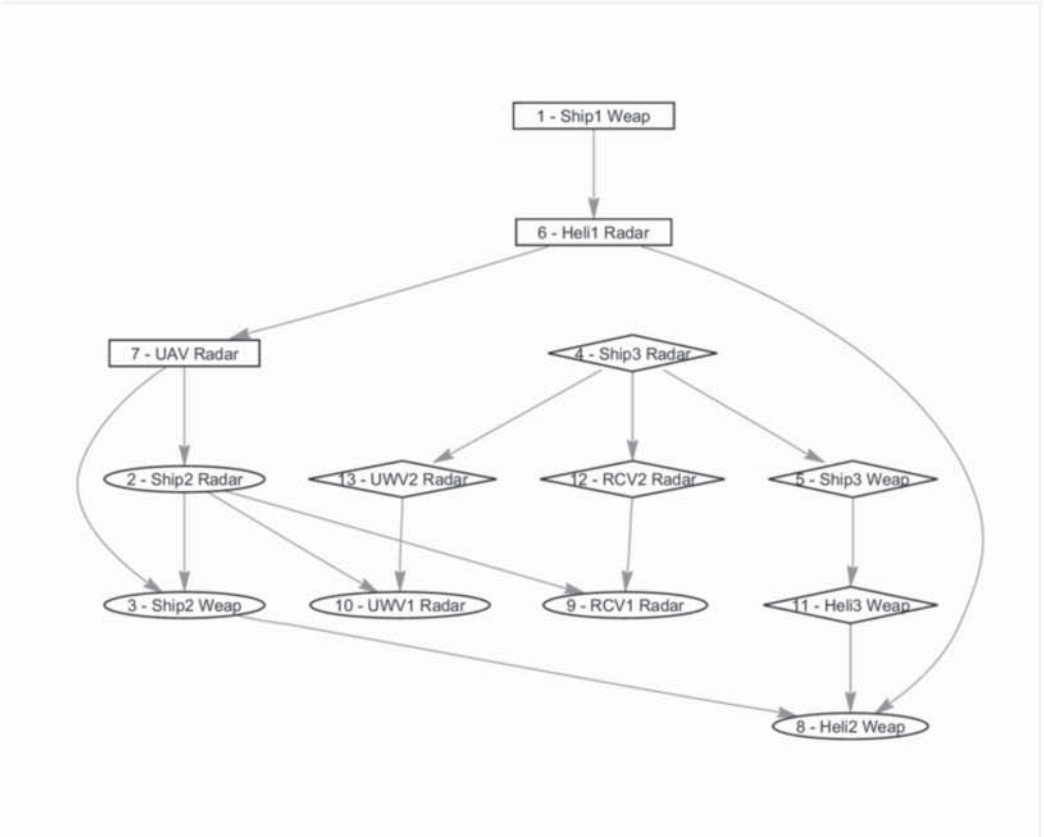


Figure 4.33. Developmental dependencies in architecture B of the Naval Warfare Scenario. Weap = weaponry. UAV = Unmanned Air Vehicle. Heli = MH60 helicopter. RCV = Remotely Controlled Vehicle. UWV = Unmanned Water Vehicle. Rectangles: surface systems. Ellipses: Anti-submarine systems. Diamonds: Anti-mine systems. Surface systems are developed first.

Architecture C

In architecture C, the surface system is developed independently from the others. The anti-mine and anti-submarine systems begin their development in parallel, but the completion of the anti-mine system requires input from the development of the anti-submarine system.

The matrices of the parameters modeling the developmental dependencies in architecture C are:

$$
{}^{dev}SOD^{C} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.15 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.15 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0.6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.55 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.65 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.65 & 0.65 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

$$
{}^{dev}COD^{C} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 60 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 55 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 45 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 45 & 35 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Figure 4.34 shows the developmental dependencies between the systems, according to the choices of the developmental manager in architecture C.

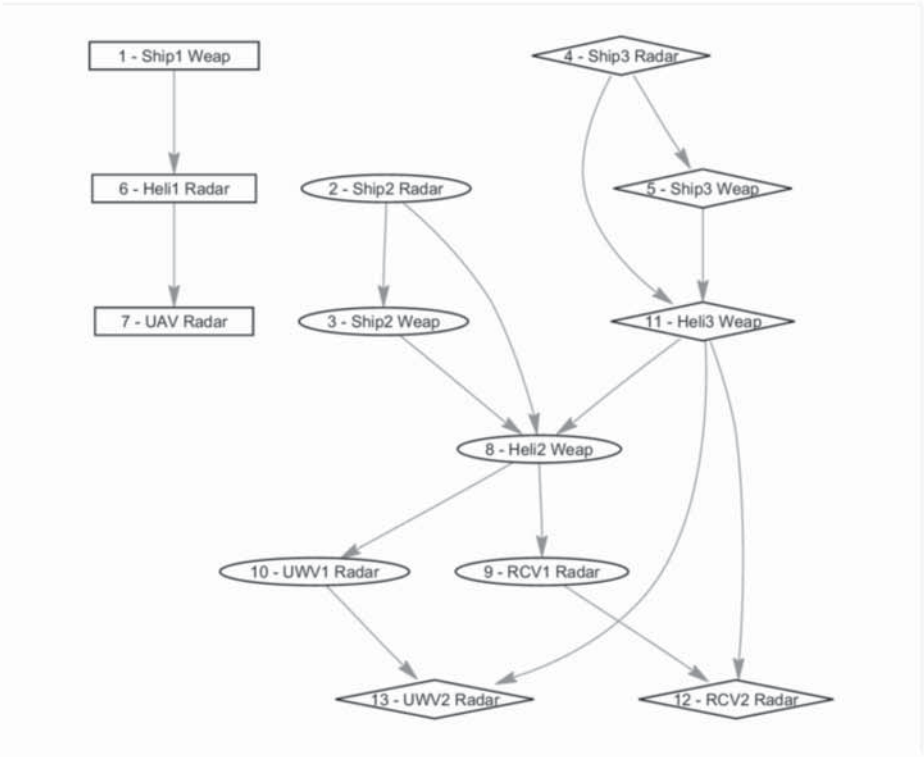


Figure 4.34. Developmental dependencies in architecture C of the Naval Warfare Scenario. Weap = weaponry. UAV = Unmanned Air Vehicle. Heli = MH60 helicopter. RCV = Remotely Controlled Vehicle. UWV = Unmanned Water Vehicle. Rectangles: surface systems. Ellipses: Anti-submarine systems. Diamonds: Anti-mine systems. Surface systems are developed independently from the other two categories.

Deterministic SDDA analysis

I first performed deterministic analysis, to compute the expected time of development, and to identify the most critical systems and dependencies in the development network. Figure 4.35 shows the basic schedule, in SDDA, SDDAmax, and PERT models, for architecture A. Figure 4.36 shows the basic schedule with SDDAmax model, for all three architecture. The schedule according to the three models, without any delay, exhibits the same aspect already underlined in the previous applications: SDDA and SDDAmax, exploiting partial dependencies, result in a schedule that allows for

early completion of the development of the SoS. SDDA, less conservative than SDDAmax, shows a longer development time for many of the component systems. This approach, that uses more resources and entails more risks, may however allow for partial recovery, when delays occur. The schedule according to SDDAmax for all three architectures shows how different component systems complete their development at different times. Therefore, different capabilities are achieved at different times, and SODA will be used to evaluate this aspect. Considerations about partial capabilities, together with criticalities, risk, and delay absorption, give support to the user decisions in development.

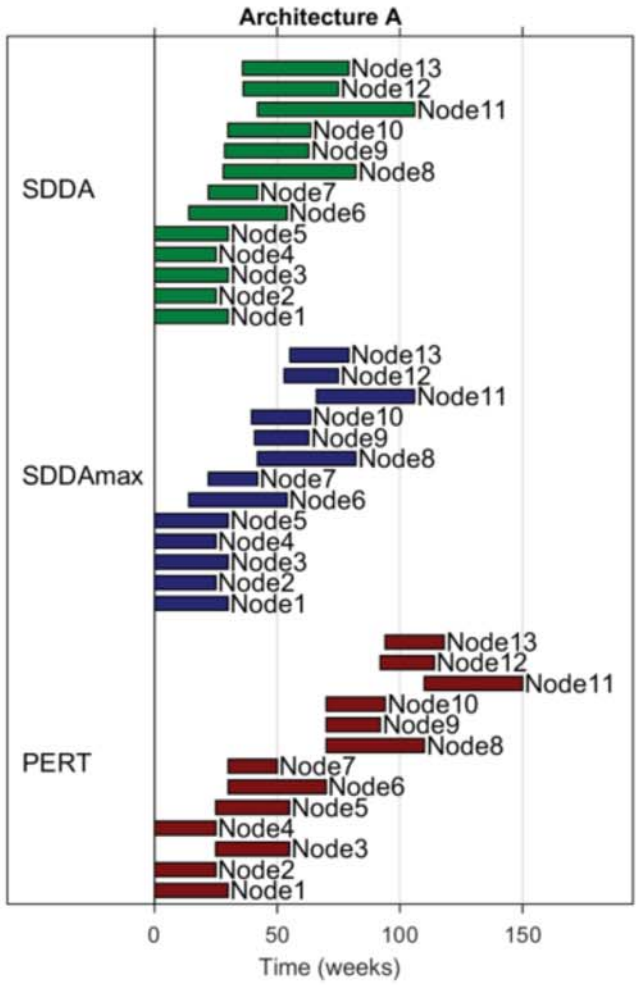


Figure 4.35. Gantt chart of the development of architecture A of the Naval Warfare Scenario, in the SDDA, SDDAmax, and PERT models.

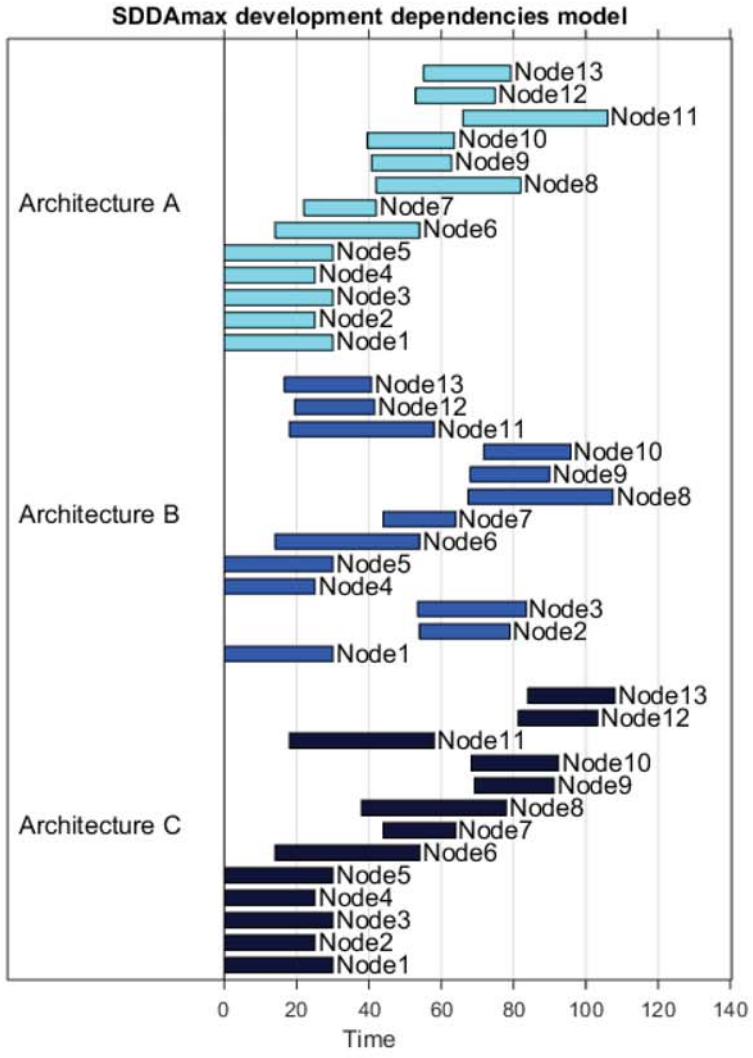


Figure 4.36. Gantt chart of the three architectures of the Naval Warfare Scenario, in the SDDAmax model.

First of all, I used deterministic analysis with punctuality of the systems at the maximum level, to compute the best time of development. I also computed the completion time of each system, useful to evaluate possible partial capabilities achieved, and the development time of each system (that is, its width on the Gantt chart), useful to evaluate the use of resources. Results are shown in table 4.12.

Table 4.12.: Deterministic SDDA analysis of the Naval Warfare Scenario. Shortest completion times of each component system in each architecture, according to the three models of dependency.

Architecture and overall time	**SDDA**		**SDDAmax**		**PERT**	
	t_C^i	t_D^i	t_C^i	t_D^i	t_C^i	t_D^i
	30	30	30	30	30	30
	25	25	25	25	25	25
	30	30	30	30	55	30
A	25	25	25	25	25	25
	30	30	30	30	55	30
SDDA: 106	54	40	54	40	70	40
	42	20	42	20	50	20
SDDAmax: 106	82	54	82	40	110	40
	62.8	34.2	62.8	22	92	22
PERT: 150	63.6	33.7	63.6	24	94	24
	106	64	106	40	150	40
	74.9	38.7	74.9	22	114	22
	79.2	43.3	79.2	24	118	24
	30	30	30	30	30	30
	79	25	79	25	115	25
	83.5	31.5	83.5	30	145	30
B	25	25	25	25	25	25
	30	30	30	30	55	30
SDDA: 107.5	54	40	54	40	70	40
	64	20	64	20	90	20

continued on next page

Table 4.12.: *continued*

Architecture and overall time	**SDDA**		**SDDAmax**		**PERT**	
	t_C^i	t_D^i	t_C^i	t_D^i	t_C^i	t_D^i
SDDAmax: 107.5	107.5	57	107.5	40	185	40
	90	40.2	90	22	137	22
PERT: 185	95.8	43.8	95.8	24	139	24
	58	40	58	40	95	40
	41.5	22	41.5	22	47	22
	40.6	24	40.6	24	49	24
	30	30	30	30	30	30
	25	25	25	25	25	25
	30	30	30	30	55	30
C	25	25	25	25	25	25
	30	30	30	30	55	30
SDDA: 108	54	40	54	40	70	40
	64	20	64	20	90	20
SDDAmax: 108	78	60.7	78	40	135	40
	91.2	22	91.2	22	157	22
PERT: 183	92.4	24	92.4	24	159	24
	58	49	58	40	95	40
	103.3	37.5	103.3	22	179	22
	108	41.2	108	24	183	24

As expected, since delays do not occur, both SDDA and SDDAmax exploit the partial dependencies at the highest possible level, and therefore achieve the same completion time. However, since SDDA is less conservative, and involves longest *lead times*, the development times of each system (and, consequently, the use of resources,

and the cost) are larger. This feature will allow for partial or total delay absorption, in case the systems will not exhibit maximum punctuality.

In the following step, I used instances of deterministic analysis with decreasing punctuality, to measure the capability of recovering delays in the three models, thanks to the use of partial dependency and *lead time*. This also allows the user to identify the most critical nodes in terms of impact on development delays. This consideration is extremely important if these systems are being developed by different managers and stakeholders. One node per instance experiences a delay, with a punctuality of 50, while the other nodes have punctuality of 100. Delay recovery is computed with respect to the initial system delay. Results are shown in table 4.13.

Table 4.13.: Deterministic SDDA analysis of the Naval Warfare Scenario with delays. Completion times and delay recovery for delays in individual nodes and in all nodes, in each architecture and according to the three models of dependency.

Delayed node	**SDDA**		**SDDAmax**		**PERT**	
	Final time	**Delay recovery (%)**	**Final time**	**Delay recovery (%)**	**Final time**	**Delay recovery (%)**
Architecture A						
1	128.3	0	128.3	0	159	0
2	110.5	40	110.5	40	150	100
3	106	100	106	100	150	100
4	106	100	106	100	150	100
5	106	100	106	100	150	100
6	116	0	128	0	160	0

continued on next page

Table 4.13.: *continued*

Delayed node	SDDA		SDDAmax		PERT	
	Final time	Delay recovery (%)	Final time	Delay recovery (%)	Final time	Delay recovery (%)
7	106	100	106	100	150	100
8	106	100	129.3	0	160	0
9	106	100	106	100	150	100
10	106	100	106	100	150	100
11	106	100	116	0	160	0
12	106	100	106	100	150	100
13	106	100	106	100	150	100
All	152.9	50	183.6	17	189	59
Architecture B						
1	129.8	0	129.8	0	194	0
2	122.5	0	140.5	0	192.5	0
3	115	16.7	132.5	0	194	0
4	114.5	6.7	114.5	6.7	185	100
5	107.5	100	107.5	100	185	100
6	125.8	0	125.8	0	195	0
7	120.8	0	120.9	0	191	0
8	107.5	100	117.5	0	195	0
9	107.5	100	107.5	100	185	100
10	107.5	100	107.5	100	185	100
11	107.5	100	107.5	100	185	100

continued on next page

Table 4.13.: *continued*

Delayed node	SDDA		SDDAmax		PERT	
	Final time	Delay recovery (%)	Final time	Delay recovery (%)	Final time	Delay recovery (%)
12	107.5	100	107.5	100	185	100
13	107.5	100	107.5	100	185	100
All	186.9	15	229.3	0	236.5	45
Architecture C						
1	108	100	108	100	183	100
2	116.5	0	116.5	0	183	100
3	108	100	109	88.9	183	100
4	140.5	0	140.5	0	190.5	0
5	117	0	129	0	192	0
6	108	100	108	100	183	100
7	108	100	108	100	183	100
8	116.7	12.7	126.7	0	193	0
9	108.3	94	116.5	0	184	80
10	111	0	118	0	186	0
11	109	90	134.7	0	193	0
12	108	100	108.3	94	184	80
13	108	100	111	0	186	0
All	161.2	43	219.8	0	225.5	55

Architecture A allows for more delay recovery, especially with the SDDA model. As expected, the SDDA model, which has the longest *lead times*, reacts better to

delays. This comes at the cost of longer development times of each system. When delays occur in critical systems, the SDDAmax model results in a completion time similar to that of PERT. It is interesting to note that, due to the different impact of the dependencies, the most critical systems are not exactly the same in different models. However, most of the critical systems are present among the different models, as expected. Table 4.14 shows the most critical systems for each architecture, and each model.

Table 4.14. Most critical systems in the development of the Naval Warfare Scenario SoS.

	SDDA	**SDDAmax**	**PERT**
Architecture A	1, 6, 2	8, 1, 6	6, 8, 11
Architecture B	1, 6, 2	2, 3, 1	6, 8, 1
Architecture C	4, 5, 8	4, 11, 5	8, 11, 5

Deterministic SDDA and SODA combined analysis

The development time of the various systems can be used in conjunction with the SODA methodology to give information about the partial capabilities attainable in the different architectures during the development. Figure 4.37 shows the operational dependencies of the capabilities of interest on the various systems.

I present here results obtained with SDDA. SDDAmax and PERT analyses can be conducted in the same way, and yield longer development times. When each system reaches 80% of its development time, it begins to contribute to the operability, according to the dependencies in the SODA model. Figures 4.38, 4.39, and 4.40 show how the capabilities of detection and engagement of the threats in the Naval Warfare Scenario are available over time during the development of the SoS in the three different architectures. In this case, no delay occurs in the development. Besides showing just the time at which partial capabilities are achieved, these figures also give

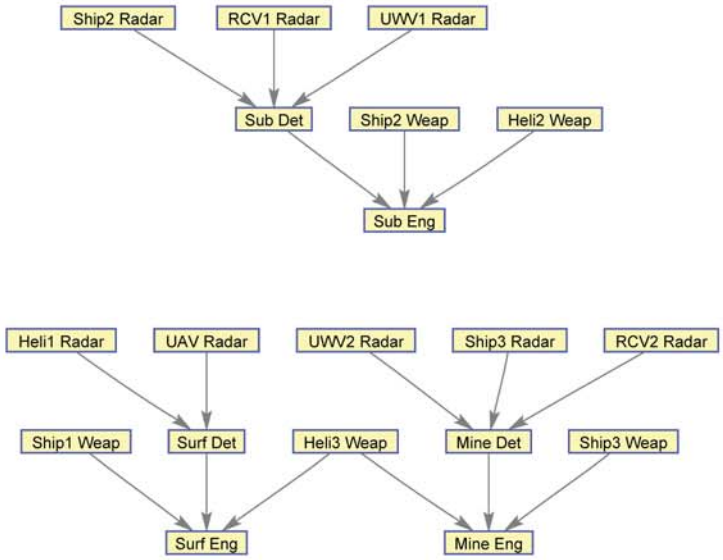


Figure 4.37. Operational dependencies of systems and capabilities in the Naval Warfare Scenario. Weap = weaponry. UAV = Unmanned Air Vehicle. Heli = MH60 helicopter. RCV = Remotely Controlled Vehicle. UWV = Unmanned Water Vehicle. Sub = Submarine. Surf = Surface. Det = Detection. Eng = Engagement.

some information about the impact of the deployment of systems and their operational dependencies. In architecture B, for example, the first radars and weapon systems deployed give a medium-high capability of engaging the enemy at $t = 45$ even if the detection capability is not fully achieved yet.

Table 4.15 shows the time required to achieve 50%, 75%, and 100% of each capability in the SoS, computed with SDDA and SODA analysis, for all three architectures.

The table shows how architecture B is slower in achieving 50% of the capability in the anti-submarine systems, but faster than the other architectures in achieving most of the full capabilities. Architecture A is the fastest in achieving partial and full capabilities in the anti-submarine systems. Architecture C is as fast as architecture B to achieve full capabilities of the surface systems, and faster in the anti-submarine systems, but slower in the development of the anti-mine systems.

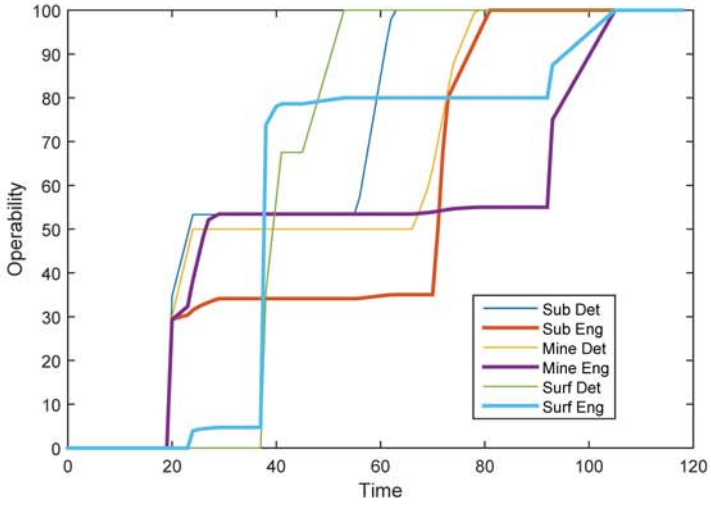


Figure 4.38. Partial capabilities in architecture A of the Naval Warfare Scenario.

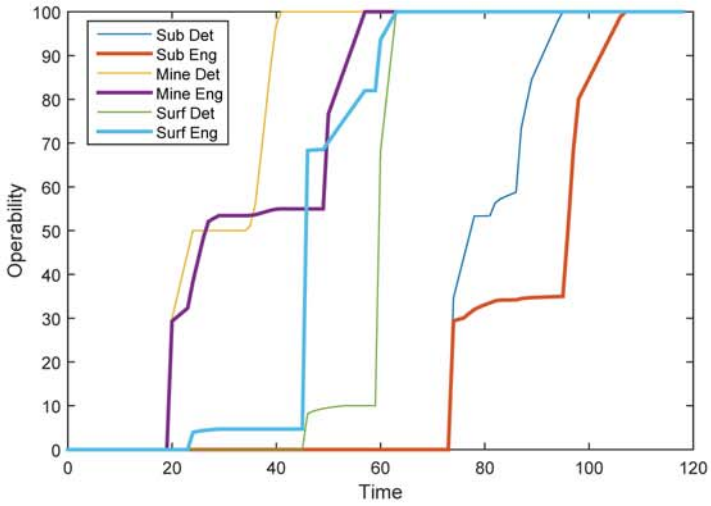


Figure 4.39. Partial capabilities in architecture B of the Naval Warfare Scenario.

When delays occur, the architectures have different reactions. Table 4.16 shows the time required to achieve 100% of the capabilities in the three architectures when delay occurs in one system.

Some delays do not impact the architectures, for example the anti-mine systems in architecture B do not suffer substantial delays in most of the cases. Architecture C,

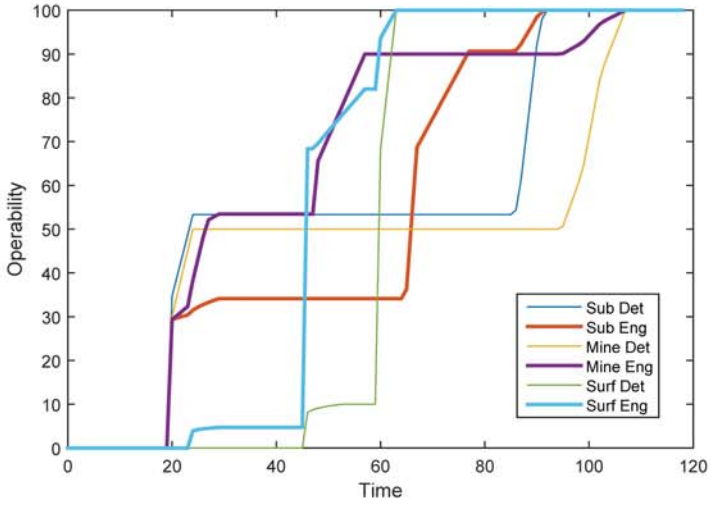


Figure 4.40. Partial capabilities in architecture C of the Naval Warfare Scenario.

Table 4.15. Most critical systems in the development of the Naval Warfare Scenario SoS.

	Architecture A			**Architecture B**			**Architecture C**		
	50%	**75%**	**100%**	**50%**	**75%**	**100%**	**50%**	**75%**	**100%**
Sub Detect	25	60	64	79	89	96	25	90	93
Sub Engage	73	74	82	97	99	108	67	71	93
Mine Detect	25	73	80	25	39	42	25	102	108
Mine Engage	28	94	106	28	51	58	28	53	108
Surf Detect	41	48	54	61	62	64	61	62	64
Surf Engage	39	40	106	47	54	64	47	53	64

instead, experiences long delays in the anti-submarine systems and anti-mine systems. Architecture A is robust to delays for what concern anti-submarine systems and detection capabilities. The systems that result more robust are those which are developed in intermediate phases of the development of the whole architecture, and have weak dependencies on the inputs from the predecessors. This allows for longer

Table 4.16. Most critical systems in the development of the Naval Warfare Scenario SoS.

Delayed system	**Architecture**	**Sub Det**	**Sub Eng**	**Mine Det**	**Mine Eng**	**Surf Det**	**Surf Eng**
Ship1 weaponry	A	86	105	102	129	77	129
	B	119	130	42	58	87	87
	C	93	93	108	108	87	87
Ship2 radar	A	64	87	80	111	54	111
	B	104	123	42	58	64	64
	C	101	101	117	117	64	64
Ship3 radar	A	64	82	80	106	54	106
	B	96	115	56	91	64	91
	C	125	125	141	141	64	91
Ship3 weap	A	64	82	80	106	54	106
	B	96	108	42	79	64	79
	C	102	102	117	117	64	67
Heli1 radar	A	74	92	90	116	64	116
	B	115	126	42	58	83	83
	C	93	93	108	108	83	83
Heli2 weapon	A	64	89	80	113	54	113
	B	96	114	42	58	64	64
	C	107	107	122	122	64	64

parallel development, leaving spare time that can be used to recover some of the delays.

Stochastic SDDA analysis and development decision support

In this section, I show an example of the use of stochastic SDDA analysis for decision support. Based on the results of the deterministic analysis, I focus on architecture A, which is the fastest to achieve partial capabilities, and robust to some delays. First of all, I performed stochastic analysis according to the model described in 3.2, with low uncertainty for nodes 1 to 5, medium uncertainty for nodes 6 to 10, and high uncertainty for systems 11 to 13. The baseline punctuality is 80 for each system.

Figure 4.41 shows the Gantt chart resulting from stochastic analysis of this baseline case, with information at time 0. The figure shows the larger uncertainty in the SDDAmax model. As usual, this model has later beginning times than SDDA, therefore it is less prone to risk in terms of excessive cost, and use of resources due to early beginning of development, followed by unexpected delays.

Table 4.17 demonstrates a possible use of SDDA stochastic analysis to support management and programming. Using architecture A, I make the following assumptions:

- The initial estimate of punctuality is equal to 80 for each system. With this estimate, and the level of uncertainty described earlier, I generate probability density of the beginning and completion times of each system.

- The initial decision for the scheduling of the beginning times can be the expected value of the beginning times resulting from the initial estimate, or the 10^{th} percentile (meaning a less risky choice of late start), or the 90^{th} percentile (meaning a more risky choice of early start).

- The manager can also decide if the programming policies will keep the initial beginning times, even when more information about the actual punctuality of the systems under development becomes available, or if the schedule will be reviewed after 30 weeks.

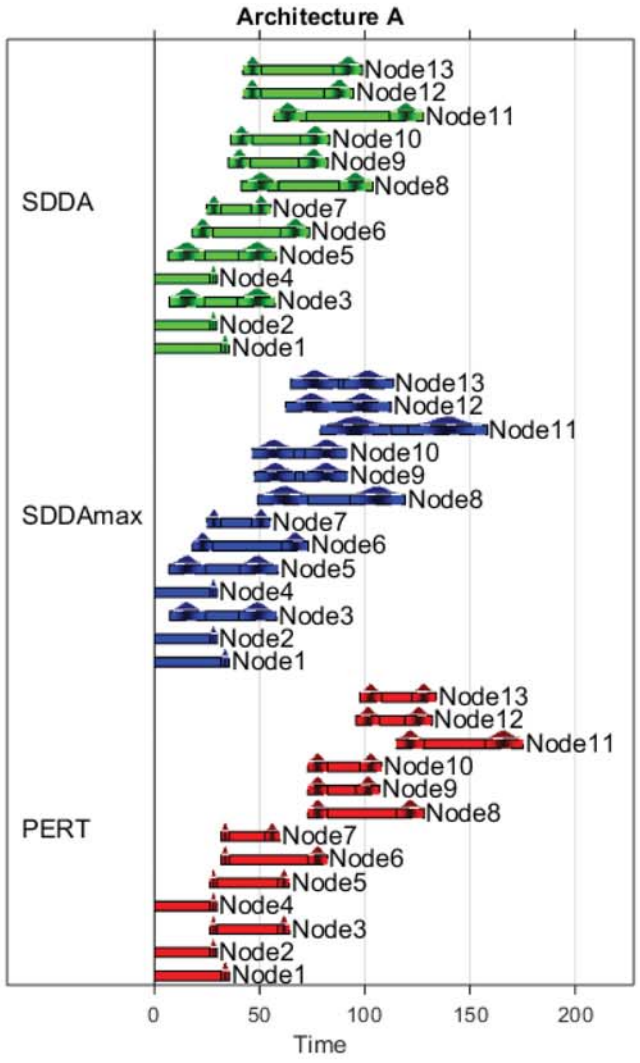


Figure 4.41. Gantt chart of the Naval Warfare Scenario SoS resulting from stochastic SDDA analysis.

- The actual punctuality of the system can be 80 (as estimated), 70, or 90. If the schedule is reviewed, information about the actual punctuality will be available.

- The minimum sum of the development time of each system is what we would have in a PERT model. Since SDDA model accounts for partial dependencies and uses a *lead time*, it generally results in longer development times of indi-

vidual systems. The actual sum of the development time of each system is used as an indicator of use of resources, that of course will impact cost.

Based on each combination of programming policies and actual punctuality of the systems, I computed the expected completion time resulting from an SDDA model having the correct information about punctuality, and the actual expected completion time resulting from the policies. This analysis has been performed over 10000 instances. I then compared the actual completion time in each case to the completion time resulting from having full information, to determine the percentage of cases in which a policy results in a longer completion time, or in a shorter completion time. I also computed the average delay in the cases where the actual completion time was higher than the case with full information, and the average gain in the cases where the actual completion time was shorter that the case with full information. Finally, I computed the minimum and actual sum of development times.

The results yield many interesting points to support management decision:

- The decision of using the expected value for the beginning times results in completion time close to that resulting from the model with correct information.
- The decision of late start causes delays with respect to the model with correct information. However, late start without schedule review also has the shortest sum of development times. Therefore, this choice can be appropriate when review of the schedule is not possible, and the most important objective is low cost and use of resources.
- The decision of early start yields the shortest completion time and the smallest delays. However, this comes at cost of longer development times.
- Reviewing the schedule when the actual punctuality is higher than the initial estimate reduces the final delay only slightly (except in the case of late start policy), but at cost of much longer sum of development times. This policy is suggested if completion time is more important than the cost due to the use of resources.

- Reviewing the schedule when the actual punctuality is smaller than the initial estimate reduces the final gain, but also the sum of development times. This policy is suggested if use of resources is more important than completion time.

- Reviewing the schedule when the actual punctuality follows the initial estimate brings both gains and delays closer to the values resulting from the model with correct information. This policy reduces the sum of development times (but also increases the percentage of late instances) if the initial decision was an early start. It increases the sum of development times (but also decreases the percentage of late instances) if the initial decision was a late start.

I repeated the same analysis using the SDDAmax model. Results are listed in table 4.18. As expected, in general SDDAmax model results in longer completion times than SDDA, but with lower sum of development times. The same trade-off occurs among different policies based on the SDDAmax model: delays can be reduced at the cost of longer sum of development times. In the case of SDDAmax model, schedule review is always suggested when the actual punctuality is better than the initial estimate, because the delays can be highly reduced at the cost of a small increase in the sum of development times. On the other side, reviewing the schedule when the actual punctuality is smaller than the initial estimate is almost never a good policy within the SDDAmax model, because only a little reduction in the percentage of late instances is possible, while the loss in gain may be substantial. Exception to this outcome is the policy of late start. In this case, reviewing the schedule when the actual punctuality is smaller than the initial estimate will increase the probability of having a gain in the completion time.

Table 4.17. Uncertainty and decisions in the Naval Warfare Scenario SoS. Architecture A, SDDA model.

Decision for t_B^i	**Actual P_i**	**Review at $t = 30$**	**Informed $E(t_C^{13})$**	**Actual $E(t_C^{13})$**	**% late inst.**	**Avg delay**	**% early inst.**	**Avg gain**	**min $\sum_{i=1}^{13} t_D^i$**	**Actual $\sum_{i=1}^{13} t_D^i$**
Expected value	**80**	**No**	119.6	119.5	46.6	1.351	53.4	1.391	409.6	481.8
		Yes	119.6	119.6	44.5	1.063	55.5	1.198	409.6	481.8
	70	**No**	131.8	121.5	0	-	100	10.304	428.4	505.5
		Yes	131.8	125.4	0	-	100	6.173	428.4	489.1
	90	**No**	112.7	117.6	100	4.942	0	-	390.8	459.5
		Yes	112.7	117.2	100	4.454	0	-	390.8	479.6
10^{th} **percentile (late start)**	**80**	**No**	119.6	124.6	98.8	5.003	1.2	0.728	409.6	473.3
		Yes	119.6	121.8	92.7	2.29	7.3	0.607	409.6	484.2
	70	**No**	131.8	126.6	1.6	0.79	98.4	5.291	428.4	496.8
		Yes	131.8	128	0.7	0.463	99.3	3.609	428.4	492.2
	90	**No**	112.7	122.7	100	10.111	0	-	390.8	451
		Yes	112.7	119.1	100	6.372	0	-	390.8	481.7
90^{th} **percentile (early start)**	**80**	**No**	119.6	117	4.7	0.534	95.3	2.769	409.6	493.7
		Yes	119.6	117.5	5.3	0.562	94.7	2.512	409.6	479.3
	70	**No**	131.8	119.1	0	-	100	12.658	428.4	517.5
		Yes	131.8	122.8	0	-	100	8.83	428.4	485.9
	90	**No**	112.7	115.2	96.3	2.683	3.7	0.521	390.8	471.3
		Yes	112.7	115.3	95.6	2.642	4.4	0.489	390.8	477.4

Table 4.18. Uncertainty and decisions in the Naval Warfare Scenario SoS. Architecture A, SDDA model.

Decision for t_B^i	**Actual P_i**	**Review at $t = 30$**	**Informed $E(t_C^{13})$**	**Actual $E(t_C^{13})$**	**% late inst.**	**Avg delay**	**% early inst.**	**Avg gain**	**min $\sum_{i=1}^{13} t_D^i$**	**Actual $\sum_{i=1}^{13} t_D^i$**
Expected value	**80**	**No**	139.1	139.1	50.1	3.957	49.9	3.96	409.6	409.9
		Yes	139.1	139.7	49.8	1.192	50.2	1.18	409.6	410.6
	70	**No**	155.4	141.1	0.1	1.321	99.9	14.336	428.4	428.5
		Yes	155.4	150.9	0	-	100	4.452	428.4	428.7
	90	**No**	124.2	137.8	99.9	13.701	0.1	1.277	390.8	392.7
		Yes	124.2	130.2	100	4.489	0	0.146	390.8	394.5
10^{th} **percentile (late start)**	**80**	**No**	139.3	149.8	98.4	10.637	1.6	1.658	409.6	409.9
		Yes	139.3	141.5	89.6	2.182	10.4	0.637	409.6	410.5
	70	**No**	155.4	151.7	21.3	2.683	78.7	5.412	428.4	428.4
		Yes	155.4	152.7	4.3	0.523	95.8	2.69	428.4	428.6
	90	**No**	124.1	148.2	100	24.129	0	-	390.8	392.7
		Yes	124.1	132	100	6.405	0	-	390.8	394.5
90^{th} **percentile (early start)**	**80**	**No**	139.2	128.7	1.6	1.653	98.5	10.744	409.6	410
		Yes	139.3	137.7	9.9	0.626	90.1	2.231	409.6	410.5
	70	**No**	155.4	130.8	0	-	100	24.593	428.4	428.9
		Yes	155.3	148.8	0	-	100	6.362	428.4	428.6
	90	**No**	124.1	127.3	76.8	4.962	23.2	2.726	390.8	392.6
		Yes	124.2	128.3	96.3	2.693	3.8	0.487	390.8	394.5

5. A CASE STUDY: MARS EXPLORATION ARCHITECTURES

"Планета есть колыбель разума, но нельзя вечно жить в колыбели."

"A planet is the cradle of mind, but one cannot live in a cradle forever."

Константин Эдуардович Циолковский
Konstantin Eduardovich Tsiolkovski
from a letter dated 1911

As mentioned in chapters 1 and 2, this research has been driven by considerations about the new needs of the space community, due to the size and complexity of current and future space missions [132, 133]. In particular, in this chapter I address the problem of Mars Exploration Architectures, to show the contribution of SODA and SDDA methodologies to some of the aspects involved into this complex problem.

Space systems engineering has many well-established procedures for designing space systems and architecting complex missions [90]. Among the challenges and needs related to Mars Exploration Architectures that are not well addressed by this procedures, the following are addressed, in whole or in part, by SODA and SDDA:

- Add top-down approach to Mars Exploration SoS architecting
  - How do systems contribute to achieving the desired capabilities?
  - What design principles can improve SoS robustness?
- Evaluate alternate architectures
  - Which architecture can improve the overall resilience?
  - What is the impact of partial or total failures, considering the dependencies?

  - What is the impact of delays and uncertainties?

- Achieve better, explicit assessment of risk
  - Which systems are the most critical?
  - How should partial capabilities evolve over time?
- Multiple stakeholders involved
  - What is the impact of stakeholder decisions on development?

5.1 The SoS approach to Mars Exploration Architectures

"Essentially, all models are wrong, but some are useful."

George Edward Pelham Box,
Empirical Model-Building and Response Surfaces, 1987

Since the uncertainties and the risks in Mars Exploration are often coupled, and the dependencies between elements in a complex combination of architecture and scenarios, possibly under constraints, is very critical, an SoS approach can be a good candidate to provide support to decision makers in the early stage of design and architecture. This approach can help answering "what-if" questions in specific scenarios and facilitates reasoning and risk analysis, providing at the same time requirements and constraints to the traditional *bottom-up* systems engineering approach. I will make use of the SoS approach developed and described by DeLaurentis [134], whose main features I describe in this section.

5.1.1 Hierarchy of the Mars Exploration Architectures

Systems-of-Systems often exhibit a hierarchy among the systems involved. Similarly to what I described for the on-orbit satellite servicing SoS (section 4.1.2), I

developed a hierarchical representation of the Mars Exploration Architecture SoS. According to the model in [134], each level of abstraction in this hierarchy is indicated with a letter of the Greek alphabet, and in the Mars Exploration Architecture SoS consists of networks of systems and their operational and developmental dependencies. The levels I used are:

- γ level

  This is the highest — and least detailed — level of abstraction. In this application, I put this level above the largest monolithic systems, and it consists only of functions and capabilities, to which systems will eventually be allocated. This level also includes policies, economic factors and stakeholder decisions relative to the functions and capabilities. Due to the low level of detail, this level is useful to compare different architectures, that might share common objectives but a substantially different decomposition at lower levels.

- β level

  In my hierarchical taxonomy for this application, this is the intermediate level of abstraction. This level contains large complex monolithic systems, like the Orion Spacecraft, the SLS, a whole control center, and communication satellites. Smaller scale functions and capabilities, related to individual systems, may also be listed in this level. Considerations about policies, economics, and stakeholders related specifically to these monolithic systems are also part of this level.

- α level

  This is the lowest — and most detailed — level of abstraction for this application. The systems in the β level are decomposed into onboard systems and facilities subsystems. Examples of entities at this level are communication antennas, solar arrays, the structure of a rocket, payload, etc. Consideration about subsystems policies, economics, and stakeholders are also part of this level.

If required by the user, this hierarchical decomposition can be expanded into lower levels. Examples of dependency networks at the various level are described in section 5.2.1.

5.1.2 The three-phase approach

The second main feature in the approach from [134] is called the three-phase approach (figure 5.1). This approach has been successfully applied to research with NASA, the Missile Defense Agency, the Federal Aviation Administration, and various industrial firms.

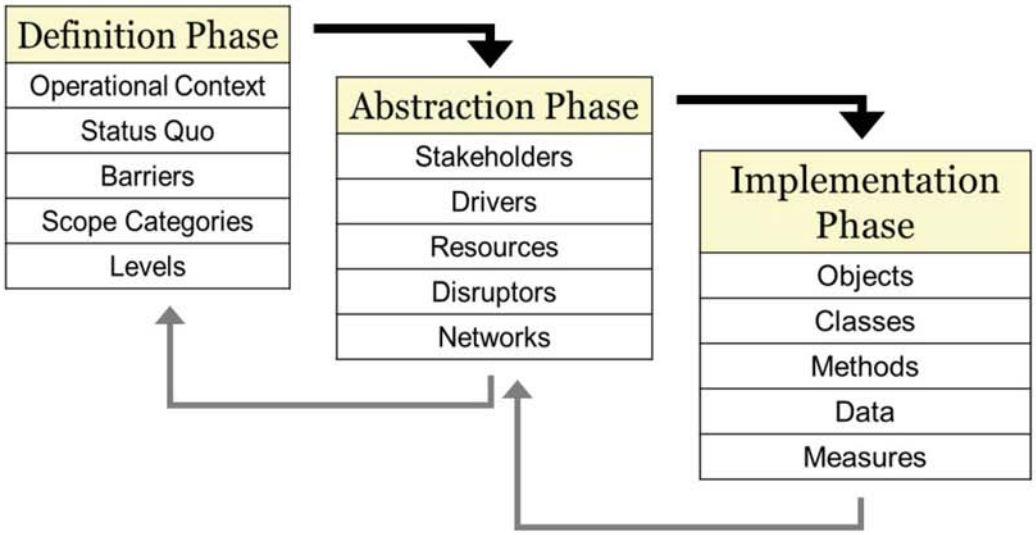


Figure 5.1. The three-phase approach for System-of-Systems analysis.

Definition phase

The definition phase is meant to increase the understanding of the underlying problem to be addressed. In this phase, the user defines the taxonomy of the systems and stakeholders involved, and the technology needs. The user also identifies the operational context, the status quo, the domain of application, time scales, de-

sired goals for future, and improved architectures. Finally, expected barriers to the preferred behavior are identified in this phase. The methodology developed in this research does not provide contribution to this phase, but rather uses part of the results of the definition phase to assess SODA and SDDA modeling parameters, to list possible questions of interest for SODA and SDDA analysis, to define uncertainties and policies that will affect the SODA and SDDA models.

In the definition phase, I drove the Resources, Operations, Policies, and Economics (ROPE) table [134] for the Mars Exploration Architecture SoS. Table 5.1 summarizes resources, operations, policies, and economics related to each level.

Table 5.1. ROPE table for the Mars Eploration Architecture SoS.

Levels	**Resources**	**Operations**	**Policies**	**Economics**
γ	High level functions, concept architectures	High level disruptions, high level drivers, goals, contractors, partners	Launch windows, timelines, partnership with commercial enterprises	Budget allocation, drivers for partners, cost of operations
β	Control centers, ground support, communication networks, in-situ resource utilization, large complex systems, system-level capabilities	Systems disruptions, systems development, contractors, partners, orbit and trajectories	Location of facilities, architectural and technological choices (e.g. split concept, in-situ resource utilization, assembly in Moon orbit, Solar Electric Propulsion), systems interactions	Budget allocation, cost of development and operation
α	On-board systems and subsystems (e.g. solar panels, controllers, structures), low level capabilities and functions	Subsystems goal and possible disruptions	Architectural and technological choices, subsystems interactions	Low level cost and budget allocation.

Choices regarding the architecture, the technology, the contractors and partners, and the goals will be reflected into the development time and above all the uncertainties both in the developmental and in the operational domain. In the case of Mars exploration, the user also needs to keep into account the launch windows. The cheapest direct trajectories, with Hohmann transfers, result in launch windows separated from each other at intervals of slightly more than two years (the synodic periodic between Earth and Mars is about 780 days). Table 5.2 list the next twelve launch windows, calculated for a trip of 6 months.

Table 5.2. Next twelve launch windows for direct launch to Mars.

Launch window number	**Date**	**Scheduled launches**
1	April 2018	InSight, Red Dragon
2	July 2020	ExoMars, Mars 2020, Mars Hope
3	September 2022	
4	November 2024	
5	January 2027	
6	February 2029	
7	April 2031	
8	June 2033	SpaceX (proposed)
9	July 2035	
10	September 2037	
11	October 2039	
12	December 2041	

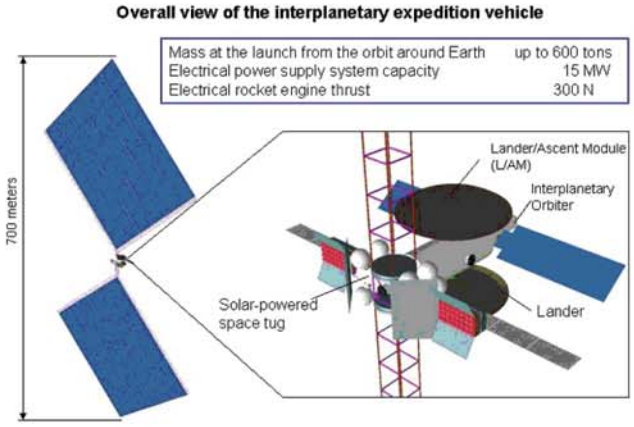
In the definition phase, I also surveyed the official concepts proposed for Mars exploration, and I found 67 official concepts proposed in the 20^{th} century, and 12 more in the 21^{st} century. Among the most recent proposals there are the NASA Design

Reference Mission 5.0 [135], the Boeing Affordable Mars Mission Design [136], and the SpaceX Mars Colonial Transporter. I sorted the concepts into two categories, with five subcategories. Each of the concepts can be implemented with different architectures, based on the systems allocated to functions and capabilities, on technological and economic constraints, and on possible sequential missions or steps required to finally achieve a mission to Mars. By architecture I mean the organization of systems and capabilities in dependency networks, accounting for stakeholders decisions, policies, economics, technology requirements, trajectories, and objectives.

1. Single mission from Earth to Mars (crew, cargo, and possible return vehicle sent in a single launch to Mars). The mission can be repeated, but each launch will carry all typology of payloads
   (a) Assembly in LEO or Moon Orbit (e.g., Project Empire)
   (b) Heavy-lift launch from Earth (e.g., Martian Piloted Complex, figure 5.2, SpaceX Mars Colonial Transporter)
2. Separate launches to Mars (multiple missions to bring separate cargo, vehicles, and crew to Mars)
   (a) Direct launches to Mars, with return (e.g., Mars Direct)
   (b) Direct launches to Mars, without return (e.g., Mars-to-stay, Mars One)
   (c) Separate launches with intermediate missions (e.g., NASA Constellation, Space Exploration Initiative, Mars Cyclers, NASA Design Reference Mission 5.0, figure 5.3)

In this application, I will focus on a single mission with heavy launch from Earth (category 1b), and a set of separate missions with intermediate objectives (category 2c). The architecture in category 1b (architecture A) will have multiple missions, each one performed with a heavy vehicle which will carry crew, cargo, equipment, and re-entry vehicle directly from Earth to Mars. The architecture in category 2c

(a) Early concept of heavy interplanetary spacecraft, with compartments for living, working, exercising, farming.



(b) Current concept of heavy interplanetary assembly.

Figure 5.2. Martian Piloted Complex. Credit RKK-Energia.

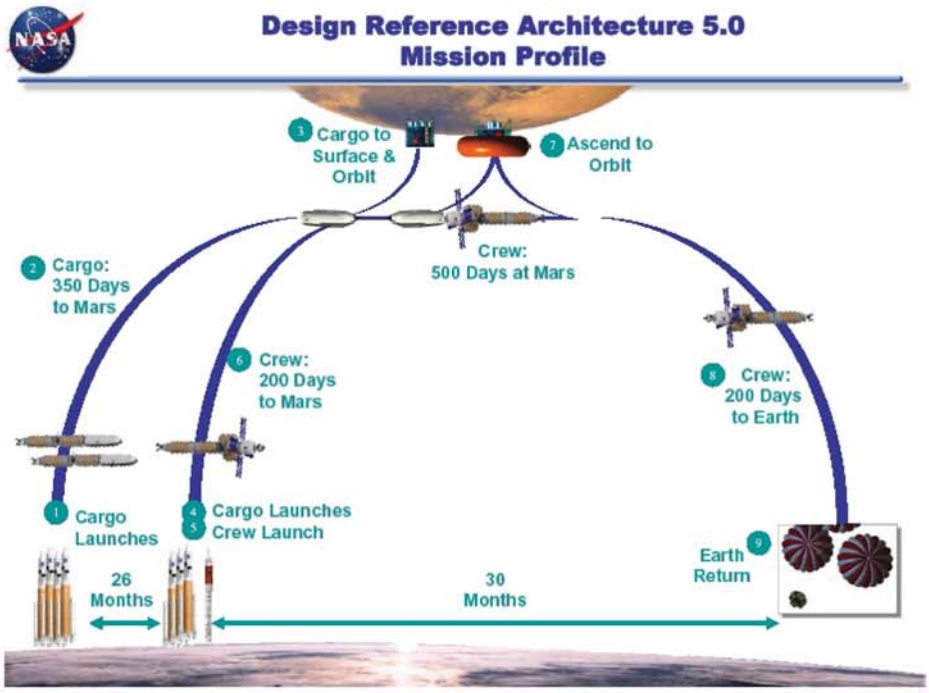


Figure 5.3. Mission profile of NASA Design Reference Mission Architecture 5.0.

(architecture B) will involve separate cargo trips to Mars or the vicinity of Mars with Solar Electric Propulsion, habitat assembly in Moon orbit, construction of bases on the Moon, crew launched to Mars with Space Launch System (SLS) rocket and Orion spacecraft (chemical propulsion), after transitioning into Moon orbit.

In anticipation of future analysis aimed at comparing more than two sample architectures, including different architectures within the same category, and among various categories, I performed a preliminary step to identify the key technologies and considerations required for each category. This step helps to assess the context and the status quo, and to guide the abstraction and implementation phases of this project.

- Common technologies across the categories
  - Control centers (locations, stakeholders, authorities)
  - Ground support (development, test, construction, integration, operation)
  - Communication networks
  - In-situ resource utilization
  - Physical and psychological impact of mission (including radiation shielding, etc.)
- Category 1a (Single missions to Mars – Assembly in LEO or Moon Orbit)
  - Rendezvous, docking, and extended — possibly automated and flexible — assembly in space
  - Medium to heavy lifters for assembly missions
  - Windows for timely launches of components and assembly missions
  - Heavy lifter
  - Chemical / hybrid / nuclear propulsion (crew is onboard)
  - Fast trajectory towards Mars

  - Piloted Entry, Descent and Landing (EDL) of super-heavy vehicle

- Category 1b (Single missions to Mars – heavy lift launch from Earth)
  - Super-heavy lifter
  - Chemical / hybrid / nuclear propulsion (crew is onboard)
  - Fast trajectory towards Mars
  - Piloted EDL of super-heavy vehicle

- Category 2a (Separate cargo and crew launch, direct launches from Earth to Mars, with return to Earth)
  - Medium to heavy lifters
  - Chemical / hybrid propulsion for crew vehicle
  - Solar Electric Propulsion / Nuclear Electric Propulsion / hybrid for cargo vehicles
  - Windows for timely launches of the components
  - Fast trajectory towards Mars for crew vehicle
  - Cheap and/or fast trajectories towards Mars for cargo and vehicles, based on type of propulsion
  - Multiple automated and accurate EDLs
  - EDL of the re-entry vehicle
  - Piloted and accurate EDL for the crew
  - In-situ production of water and fuel
  - Windows for re-entry

- Category 2b (Separate cargo and crew launch, direct launches from Earth to Mars, without return to Earth)
  - Medium to heavy lifters

  - Chemical / hybrid propulsion for crew vehicle
  - Solar Electric Propulsion / Nuclear Electric Propulsion / hybrid for cargo vehicles
  - Windows for timely launches of the components
  - Fast trajectory towards Mars for crew vehicle
  - Cheap and/or fast trajectories towards Mars for cargo and vehicles, based on type of propulsion
  - Multiple automated and accurate EDLs
  - Piloted and accurate EDL for the crew
  - Extended structures and in-situ resources utilization for colonization

- Category 2c (Separate cargo and crew launches, intermediate missions with assembly in space or on other bodies, with return to Earth)
  - Assembly in space, or on other bodies
  - Sequential mission launch windows and trajectories
  - Medium to heavy lifters
  - Chemical / hybrid propulsion for crew vehicle
  - Solar Electric Propulsion / Nuclear Electric Propulsion / hybrid for cargo vehicles
  - Windows for timely launches of the components
  - For cyclers, development of cycler vehicle, and high-velocity rendezvous
  - Fast trajectory towards Mars for crew vehicle from different origin (Earth, Moon, Near Earth Objects, Phobos and Deimos, Cyclers)
  - Cheap and/or fast trajectories towards Mars for cargo and vehicles, based on type of propulsion and origin
  - Multiple automated and accurate EDLs

- EDL for re-entry vehicle
- Piloted and accurate EDL for crew
- In-situ production of water and fuel
- Windows for re-entry
- Extended structures and in-situ resources utilization for colonization

Abstraction phase

In the abstraction phase, the user focuses on framing the problem, in order to facilitate the transition from definition to modeling and implementation. In this phase the user identifies the inter-relations among all involved entities, draws dependency networks, and ascertain the big-picture dynamics and behavior of the whole system. SODA and SDDA contributes to this phase with representation of operational and developmental dependencies at various level of the hierarchical SoS. I describe the contribution of SODA and SDDA to the abstraction phase of the Mars Exploration Architectures in section 5.2.1.

Implementation phase

In the implementation phase, the user performs the actual modeling and analysis of the whole SoS. In the implementation phase, the user explores the behavior, dynamics, and features of all the items involved, and interprets the result of the analysis. The analysis in the implementation phase has a twofold goal:

- Identifying features of interest, and increasing the knowledge about the systems, including possible unexpected, emergent behavior.
- Identifying possible ways of improving the systems while achieving the desired goals.

To show the contribution of SODA and SDDA to the implementation phase of the Mars Exploration Architectures, I instantiated part of the items identified in the abstraction phase as SODA and SDDA networks, and addressed various "what-if" questions, to test and verify the model. The questions and the results of the analysis guide decision-making in designing the Mars Exploration Architectures. The implementation phase, executed in early phase of the design process, may need to iterate back to the definition and abstraction phase for updates or verification. Final results of the implementation phase will provide requirements and constraints to the design of individual systems, executed according to traditional breakdown of systems and bottom-up approach. The high-level, top-down analysis can be repeated later during the design process, in an iterative fashion alternating holistic and reductionistic considerations.

5.2 SODA and SDDA contribution to the Mars Exploration architectures problem

"We set sail on this new sea because there is new knowledge to be gained, and new rights to be won, and they must be won and used for the progress of all people. [...] There is no strife, no prejudice, no national conflict in outer space as yet. Its hazards are hostile to us all. Its conquest deserves the best of all mankind, and its opportunity for peaceful cooperation may never come again. But why, some say, the Moon? [...] We choose to go to the Moon in this decade and do the other things, not because they are easy, but because they are hard; because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one we intend to win."

John Fitzgerald Kennedy,
from the speech at Rice University, 12 September 1962

5.2.1 Abstraction with SODA and SDDA

Developmental and Operational networks

γ level In this application, the nodes of the networks of operational and developmental dependencies for SODA and SDDA analysis at this level are capabilities and overarching goals of the architectures for Mars exploration. The developmental and sequencing dependencies at this level are of utmost importance. However, in the implementation phase, I will not characterize the nodes at the γ level with their own development time, instead I will evaluate their sequencing according to the development time and dependencies of the corresponding β level structure. I will implement the operational dependencies represented at this level as operational dependencies among nodes of interest at the corresponding β level structure, similarly to what I did in section 4.1.2.

The outcomes that the user wish to analyze, and methodologies like the Design Structure Matrices [45] can support the decision of nodes to include at the various levels. Figure 5.4 shows the developmental and operational dependencies between the capabilities at the γ level in architecture A. Figure 5.5 shows the developmental

and operational dependencies between the capabilities at the γ level in architecture B. Architecture B, which belongs to the category of concepts based on a series of intermediate steps to reach Mars, has more capabilities that need to be developed. However, this is not enough to decide that architecture A is preferable, since the capabilities in architecture A may require more effort in term of time and cost, and may result less robust and less flexible.

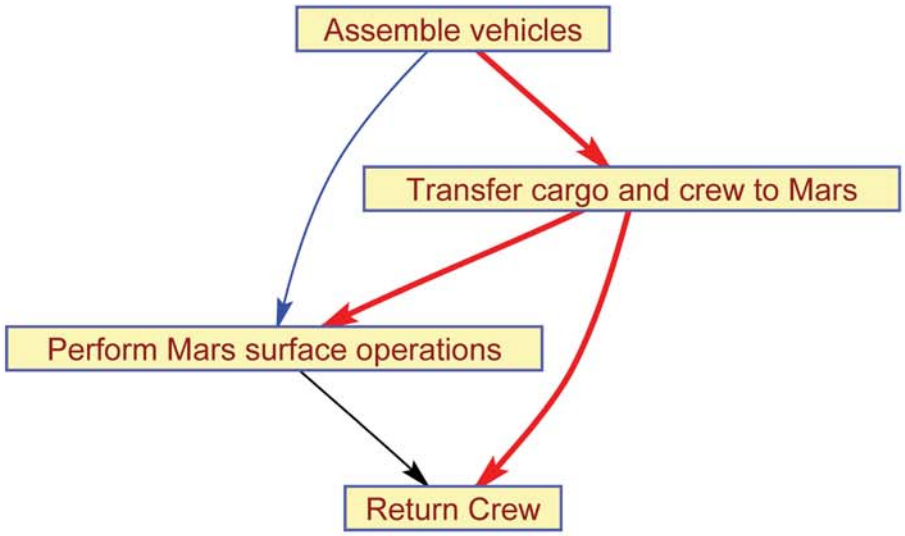


Figure 5.4. Developmental and operational dependencies at the γ level in architecture A. Thin black edges: developmental dependencies. Thin blue edges: operational dependencies. Thick red edges: both kind of dependencies.

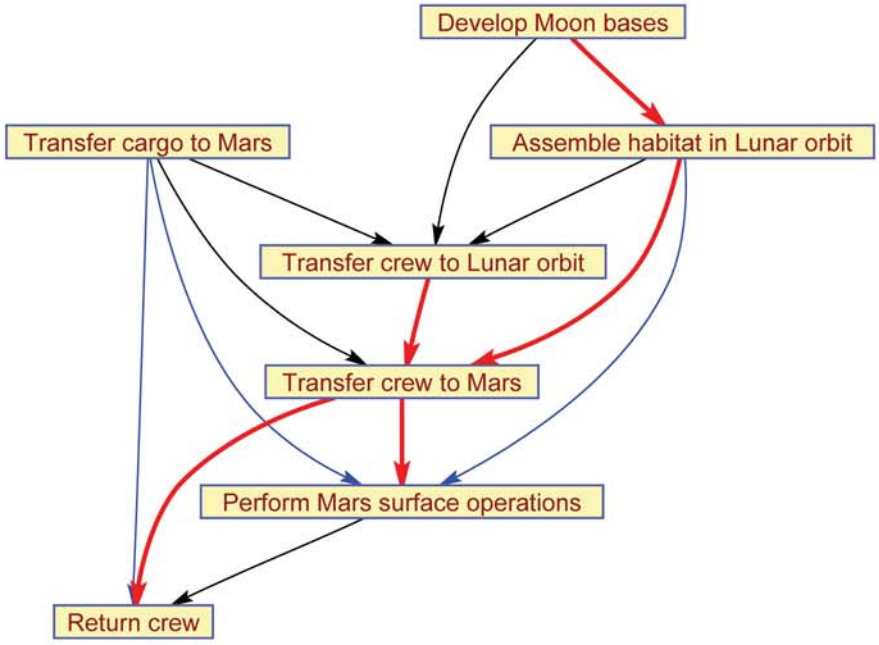


Figure 5.5. Developmental and operational dependencies at the γ level in architecture B. Thin black edges: developmental dependencies. Thin blue edges: operational dependencies. Thick red edges: both kind of dependencies.

β level In this application, the networks of operational and developmental dependencies at this level are the core of abstraction and implementation phase. For the two architectures under study, I identified operational and developmental dependencies. Figure 5.6 shows the complex network of developmental and operational dependencies between systems and capabilities at the β level in architecture A. Figure 5.7 shows only the operational dependencies. These dependencies will be used in the implementation phase to evaluate the impact of failures on the capabilities of interest. Figure 5.8 shows only the developmental dependencies. In this case, I included both dependencies due to required inputs and dependencies due to sequencing, with some dummy nodes. For example, once the crew reentry vehicle is fully developed, it does not immediately result in the partial capability of reentry, because the operational dependency of the reentry on the crew reentry vehicle takes place only after the cargo

and crew have been deployed on Mars and have executed their tasks. Therefore, in this network the capability of cargo delivery to Mars has a developmental dependency on the completion of the crew reentry vehicle, and the capability of reentry had a developmental dependency on the cargo delivery to Mars.

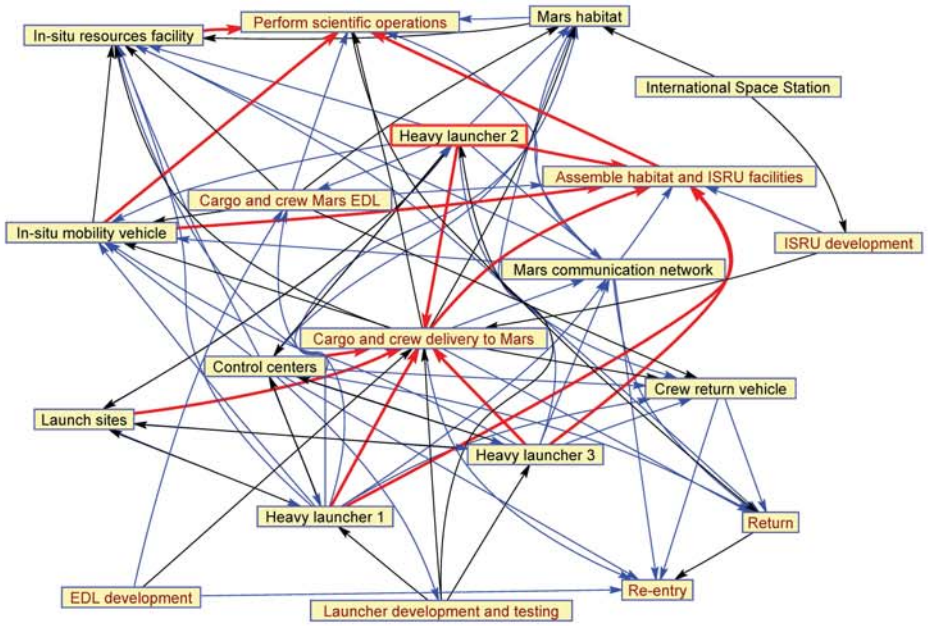


Figure 5.6. Developmental and operational dependencies at the β level in architecture A. Thin black edges: developmental dependencies. Thin blue edges: operational dependencies. Thick red edges: both kind of dependencies. Black nodes: systems. Red nodes: capabilities.

Figures 5.9, 5.10, and 5.11 show analogous dependencies for architecture B.

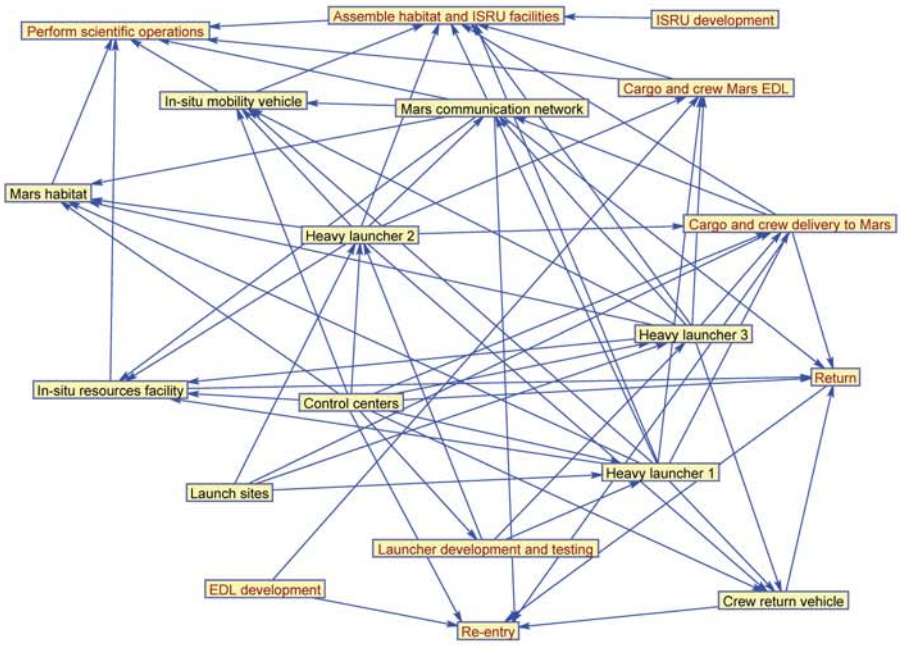


Figure 5.7. Operational dependencies at the β level in architecture A.

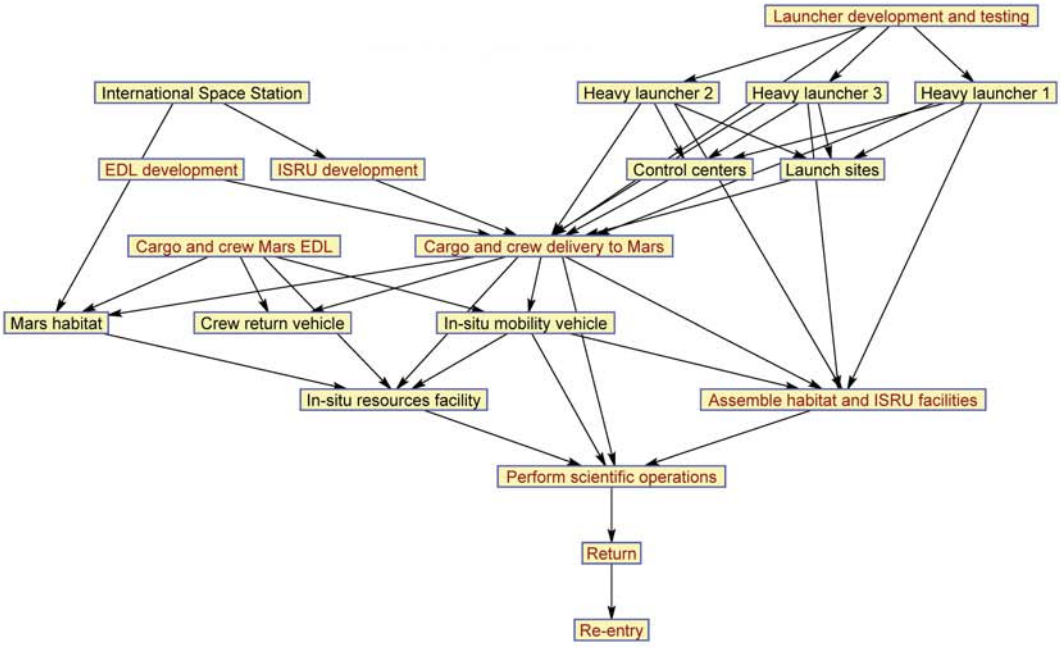


Figure 5.8. Developmental dependencies at the β level in architecture A.

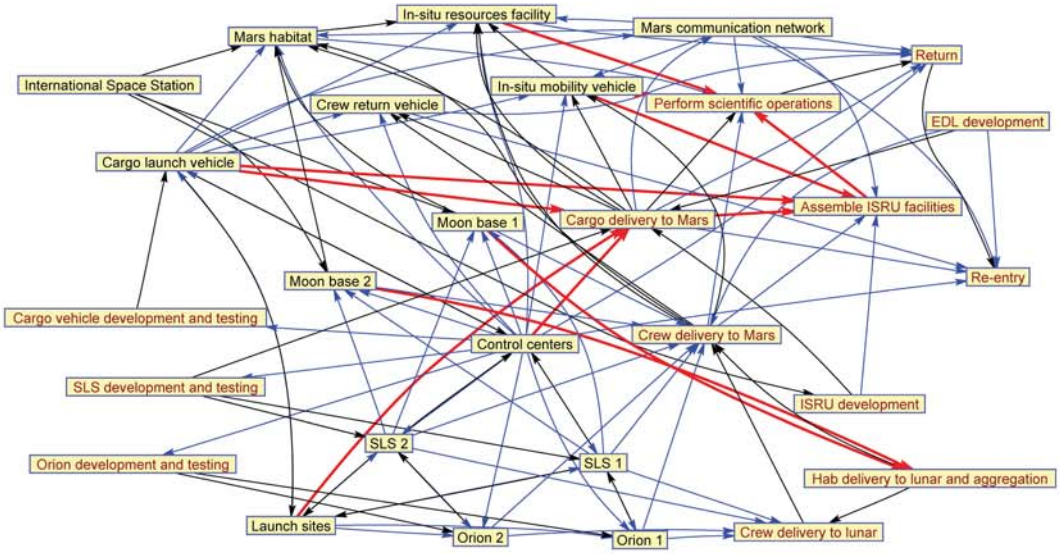


Figure 5.9. Developmental and operational dependencies at the β level in architecture B. Thin black edges: developmental dependencies. Thin blue edges: operational dependencies. Thick red edges: both kind of dependencies. Black nodes: systems. Red nodes: capabilities.

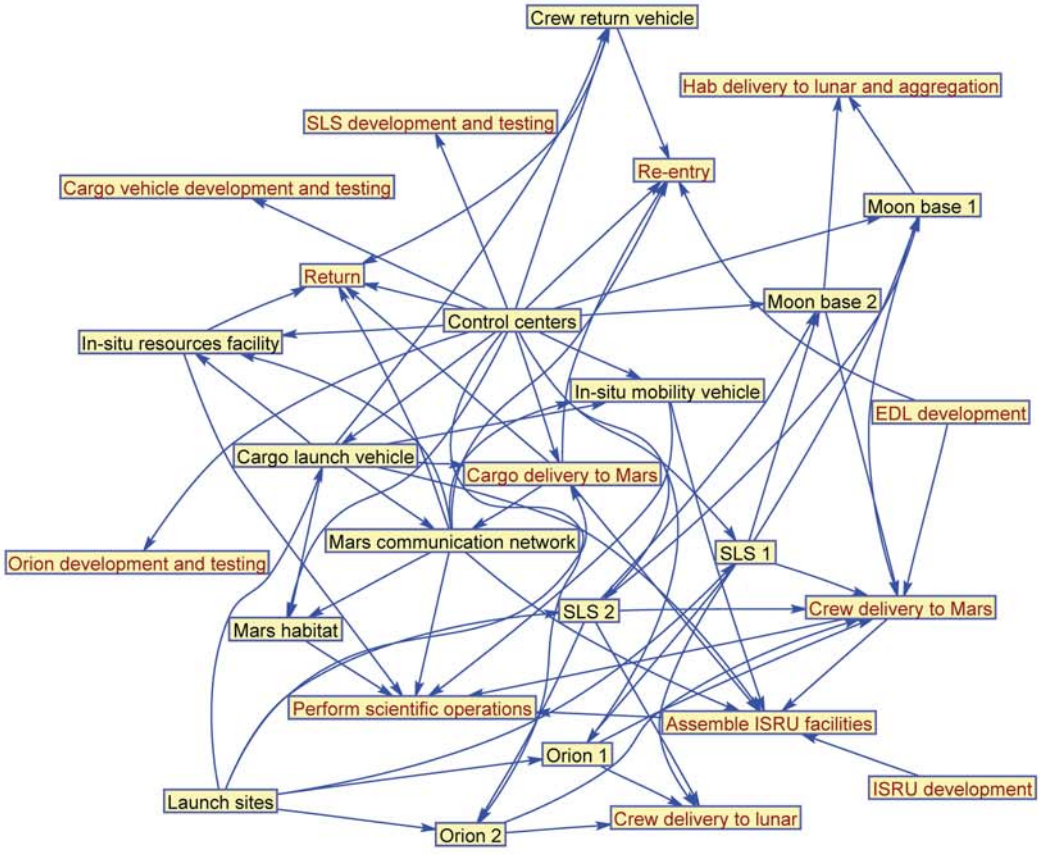


Figure 5.10. Operational dependencies at the β level in architecture B.

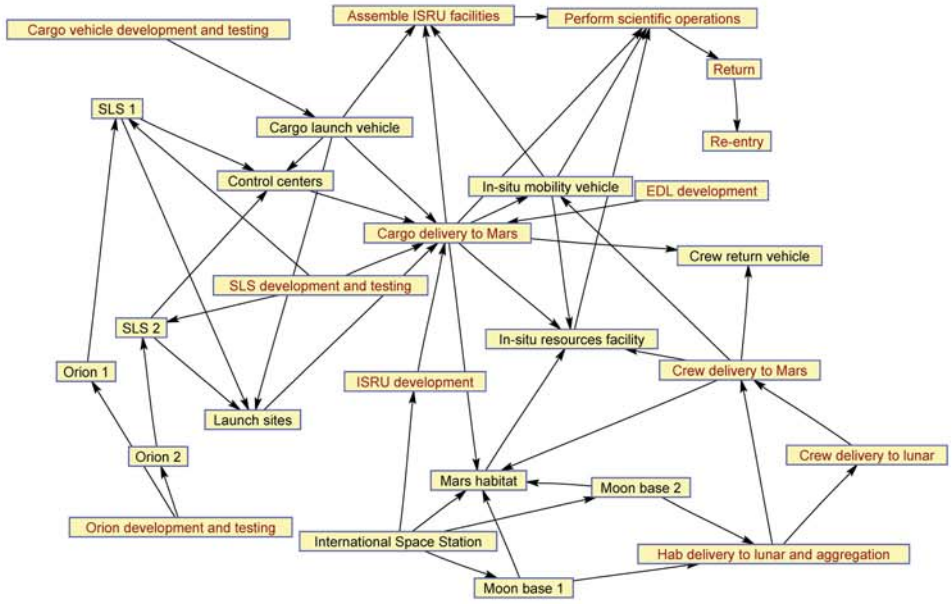


Figure 5.11. Developmental dependencies at the β level in architecture B.

α level Similarly to what described in some of the applications in chapter 4, I decomposed some of the systems at the β level into subsystems and development phases, identifying the operational and developmental dependencies. Due to the scope of this case study, I did not perform SODA and SDDA analysis at this level in the implementation phase, which would be similar to what already illustrated in chapter 4. However, to give an example of how to execute the abstraction step for this particular case study at the α level, I included some dependency networks at the α level. Figures 5.12 shows the operational dependencies among the subsystems of Orion spacecraft, similar to the satellite subsystems from the application in 4.1.2. Figure 5.13 shows an example of the dependencies among stages of the development of the Space Launch System. Figure 5.14 illustrates an example of developmental dependencies of the systems of a cargo transportation vehicle developed in part by commercial contractors.

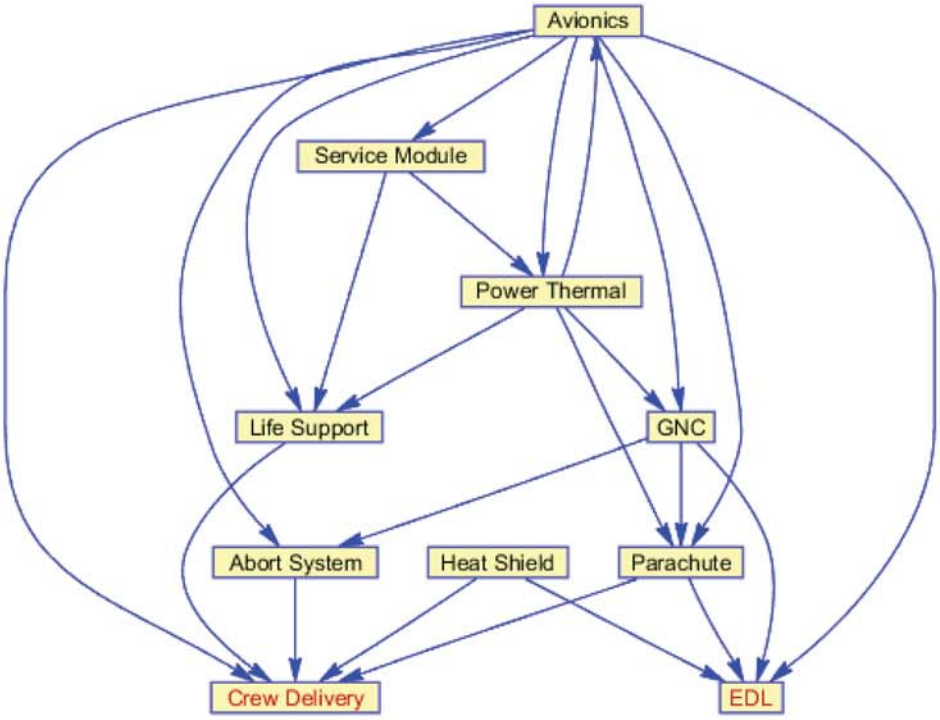


Figure 5.12. Operational dependencies at the α level of a manned spacecraft. Credit: William O'Neill.

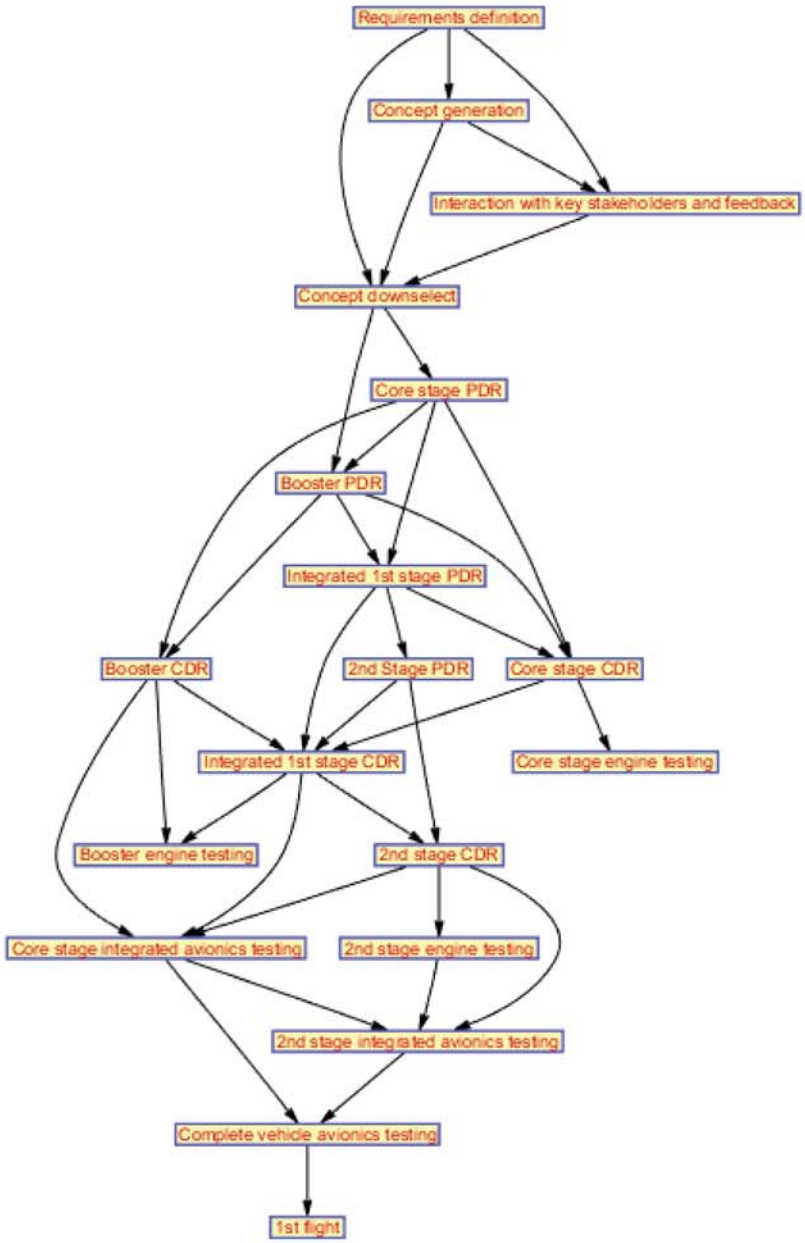


Figure 5.13. Developmental dependencies at the α level of the Space Launch System. Credit: Ashwati Das-Stuart.

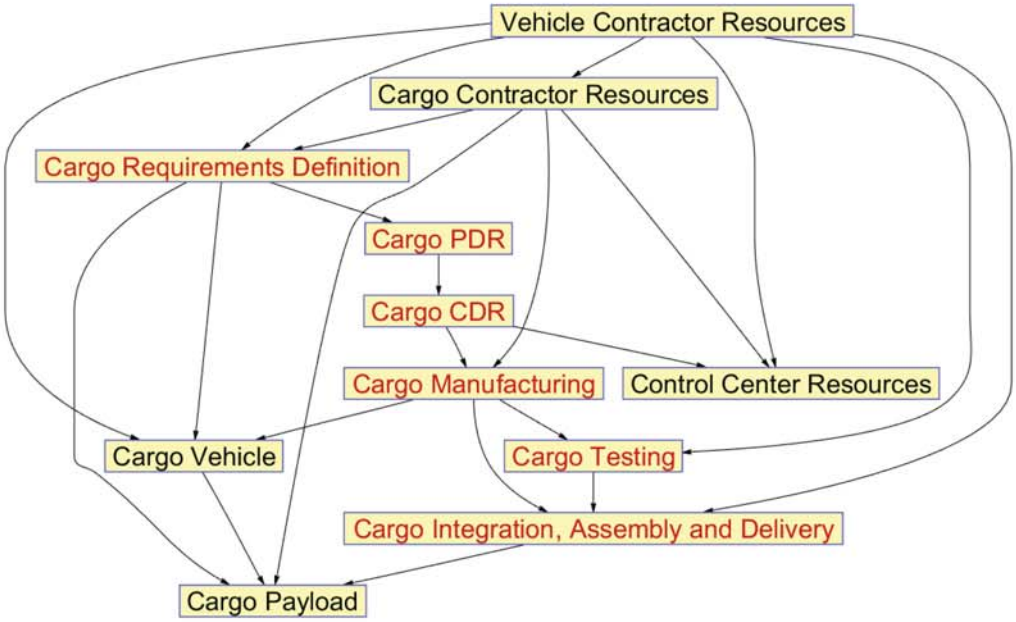


Figure 5.14. Developmental dependencies at the α level of a cargo transportation vehicle, developed in part by a commercial contractor.

5.2.2 Implementation with SODA and SDDA

"What-ifs" and SODA analysis

The following are sample questions that can be addressed with SODA analysis for the Mars Exploration Architecture SoS:

1. What are the most critical systems for the operability of the whole mission?
2. What is the loss in capability of performing scientific operations of Mars if one facility for in-situ resources production experiences a major failure?
3. What is the robustness of a given architecture?

The SOD, COD, and IOD of the operational dependencies reflect the technological requirements of the involved systems and capabilities.

I first evaluated the losses in operational capabilities due to major failures in various systems. I modeled the SE of the failed system with a Beta(2,11) probability density function, while the other systems are fully operational. Table 5.3 shows the results of this SODA stochastic analysis for single failures in architecture A. Launch sites are critical for the operability of carrying crew to Mars, while failures in one of the launchers result less critical. Failures in the in-situ mobility vehicle are critical to performing science operations, while failures in the in-situ resource facility can impact also the capability of re-entry on Earth, in part based on production on Mars. Table 5.4 shows the results of the same kind of analysis for single failures in architecture B, where the increased redundancy of the systems and the strategy with intermediate steps reduces the impact of failures.

Table 5.3. Stochastic analysis of the impact of failures on the overall operability in architecture A.

System with failure	**O(crew on Mars)**	**O(science operations)**	**O(return)**
Launch sites	14.8	74.3	37.3
Heavy launcher 1	95.5	95.4	96.1
In-situ resource facility	100	96.0	91.4
In-situ mobility vehicle	100	89.2	100
Crew return vehicle	100	100	83.7

Table 5.4. Stochastic analysis of the impact of failures on the overall operability in architecture B.

System with failure	**O(crew on Mars)**	**O(science operations)**	**O(return)**
Launch sites	79.7	71.9	51.8
SLS 1	93.2	99.9	100
Orion 1	94.3	99.2	100
Moon base 1	97.1	99.6	100
In-site resource facility	100	96.9	93.4
In-situ mobility vehicle	100	91.2	100
Crew return vehicle	100	100	97.8

I computed the robustness of the two architecture to multiple failures in the systems (modeled with Beta(2,11) for each system), using the deployment of cargo and

crew on Mars, scientific operations, and return as nodes of interest. The values, computed according to equation 4.2, are as follow.

$$Rob_A = 0.39$$

$$Rob_B = 0.37$$

"What-ifs" and SDDA analysis

The following are sample questions that can be addressed with SDDA analysis for the Mars Exploration Architecture SoS:

1. What is the delay (including possible missed launch windows) and impact on capabilities if a launcher experiences a delay during development?
2. What are the most critical systems in terms of development?
3. What if the initial assessment of development time is wrong? Which development policy is the most flexible and the safest?

In this section I address the first two sample questions. The third can be answered with the kind of analysis I performed at the end of section 4.3.1.

The SOD and COD of the developmental dependencies, and the minimum and maximum development time reflect the complexity and the Technology Readiness Level (TRL) of the systems involved. Launch windows occur at a distance of 110 weeks from one another, therefore uncertainties and delays can have a higher impact, because the resulting architecture may miss a launch window.

Figure 5.15 shows the baseline schedule resulting from the developmental dependencies for architecture A, and figure 5.16 shows the baseline schedule resulting from the developmental dependencies for architecture B. Due to launch windows, dependencies and times of development, both architecture show a return at the same time in their baseline schedule (lasting about 12 years). Architecture A appears to have

more flexibility and spare time before the first launch, so a possible capability to absorb more delays. However, it is also the architecture whose systems are expected to have higher impact, due to their size and to the absence of intermediate steps.

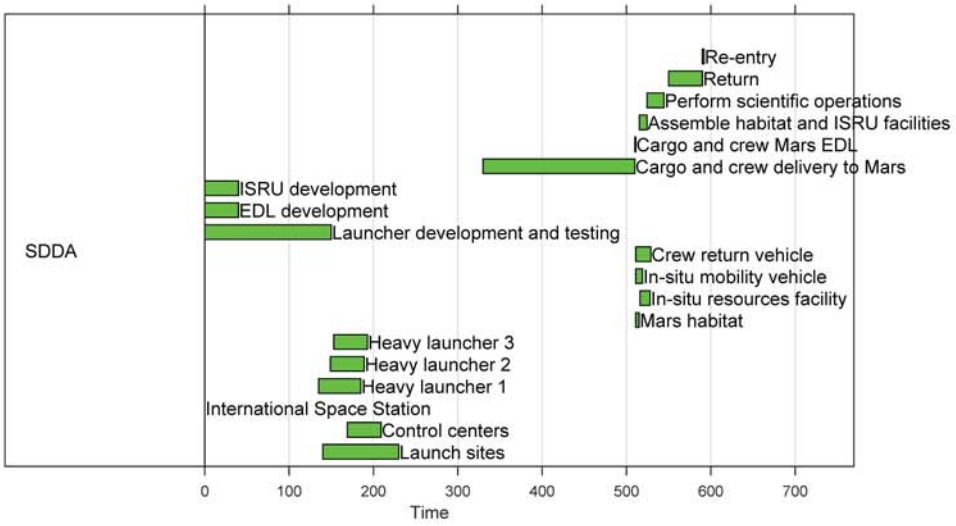


Figure 5.15. Baseline schedule of the β level of architecture A for Mars exploration.

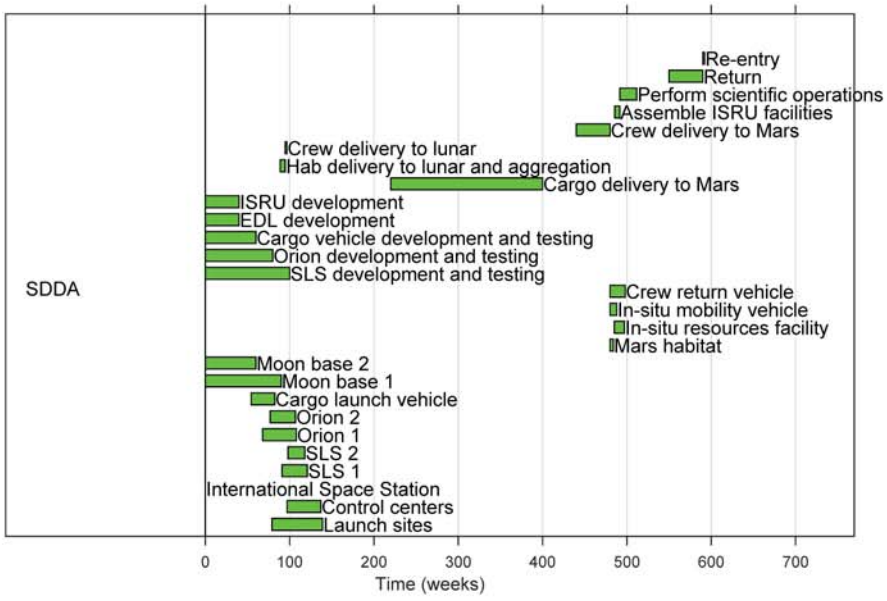


Figure 5.16. Baseline schedule of the β level of architecture B for Mars exploration.

Deterministic SDDA analysis suggests the criticality of the systems in the developmental domain. Table 5.5 shows the effect of delays in individual systems or development stages on the overall schedule in architecture A. In this case, I used a punctuality of 10 for the delayed systems. Due to the discrete launch windows, large groups of systems cause the same impact on the final schedule in this deterministic case. The development and testing of new, heavy launchers results more critical than the final phase of building the launchers. Delays in assembling the habitat and ISRU facilities on Mars have delays on the execution of scientific research, but they are still absorbed before the launch window for return.

Table 5.6 shows results of the same kind of analysis in architecture B. Due to the lower amount of spare time before the first launch in this architecture, I expected it to have less robustness. However, since the development of the various technologies can proceed in parallel, and the uncertainty has less impact than in the case of heavy launchers, the architecture is robust for all delays in single systems or developmental stages.

Table 5.5. Deterministic analysis of the impact of delays on the overall schedule in architecture A.

System with delay	$t_C^{crewonMars}$	$t_C^{scienceoperations}$	t_C^{return}
None	510	544.4	592
Launch sites	510	544.4	592
Heavy launcher 1	510	544.4	592
Heavy launcher 3	510	544.4	592
Crew return vehicle	510	544.4	592
Launcher development and testing	620	654.4	702
ISRU development	510	544.4	592
Assemble habitat and ISRU facilities	510	549.3	592

Table 5.6. Deterministic analysis of the impact of delays on the overall schedule in architecture B.

System with delay	$t_C^{crewonMars}$	$t_C^{scienceoperations}$	t_C^{return}
None	480	511.6	592
Launch sites	480	511.6	592
SLS 1	480	511.6	592
Orion 1	480	511.6	592
SLS development and testing	480	511.6	592
ISRU development	480	511.6	592
Hab delivery to lunar and integration	480	511.6	592
Assemble ISRU facilities	480	518.8	592

This deterministic analysis, however, puts all the systems at the same level of disrupted punctuality, which is usually not the case. I used stochastic analysis, with levels of uncertainty based on the complexity and Technology Readiness Level of each system and stage, to quantify the probability that each architecture will follow the expected schedule. For this case, I used 10000 runs of stochastic SDDA analysis. Figure 5.17 shows the expected schedule of architecture A under uncertainty. Due to the complexity of the systems involved, and the absence of intermediate steps, the uncertainty is large, and causes delays beyond the first available launch window in many cases. Figure 5.18 shows the expected schedule of architecture B under uncertainty. In this case, the uncertainty of the schedule of individual systems and stages is more variable, resulting in a lower number of instances with delays, but some of the delays can miss two or even three launch windows.

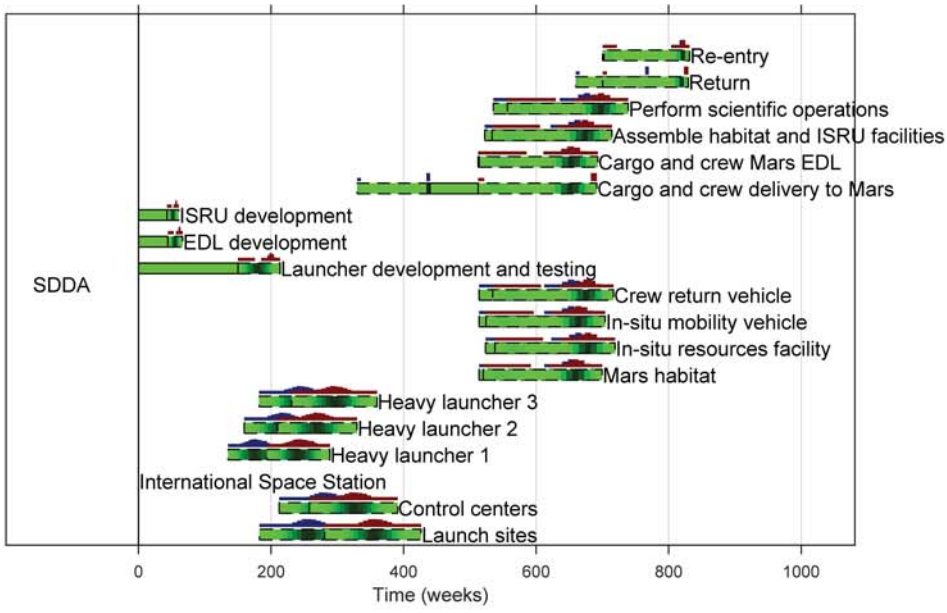


Figure 5.17. Schedule of the β level of architecture A for Mars exploration under uncertainty. Blue rectangles indicate instances of beginning time. Red rectangles indicate instances of completion time. Shadowed areas in the green Gantt chart elements indicate areas with higher probability.

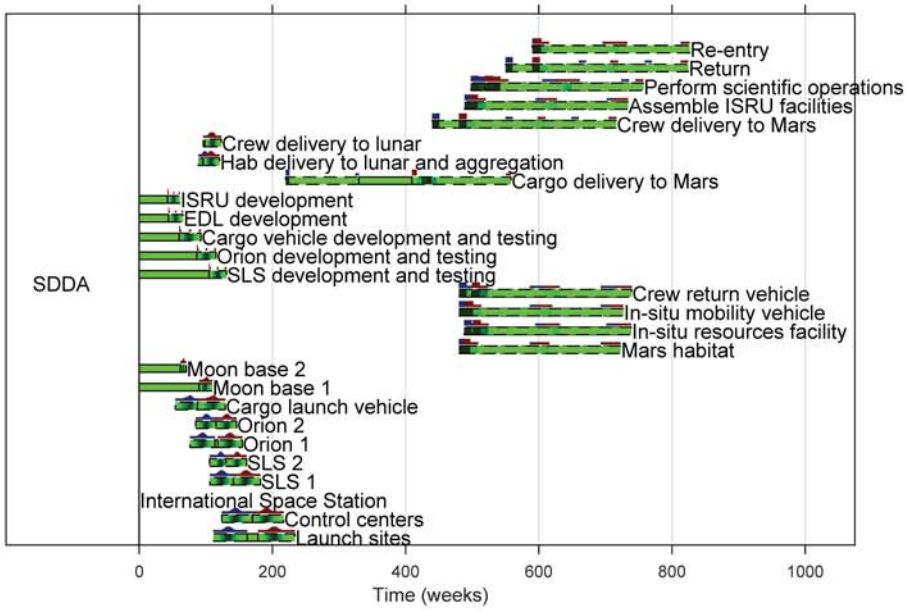


Figure 5.18. Schedule of the β level of architecture B for Mars exploration under uncertainty. Blue rectangles indicate instances of beginning time. Red rectangles indicate instances of completion time. Shadowed areas in the green Gantt chart elements indicate areas with higher probability.

Tables 5.7, 5.8, and 5.9 shows percentage of instances that completed tasks within a given time threshold in the two architectures.

Table 5.7. Percentage of instances completing delivery of crew on Mars within given time thresholds.

	Crew arrival on Mars					
	$t_C < 500$	$500 \leq t_C < 550$	$550 \leq t_C < 600$	$600 \leq t_C < 650$	$650 \leq t_C < 700$	$700 \leq t_C$
Architecture A	0 %	7.2 %	2.7 %	36.5 %	53.6 %	0 %
Architecture B	81.1 %	0.01 %	11.2 %	7.4 %	0 %	0.3 %

Table 5.8. Percentage of instances completing scientific portion of the mission on Mars within given time thresholds.

	Perform scientific operations					
	$t_C < 550$	$550 \leq t_C < 600$	$600 \leq t_C < 650$	$650 \leq t_C < 700$	$700 \leq t_C < 750$	$750 \leq t_C$
Architecture A	0 %	8.2 %	1.7 %	51.5 %	38.6 %	0 %
Architecture B	81.2 %	0 %	18.5 %	0.07 %	0.07 %	0.2 %

Table 5.9. Percentage of instances completing the return from Mars within given time thresholds.

	Return					
	$t_C < 600$	$600 \leq t_C < 650$	$650 \leq t_C < 700$	$700 \leq t_C < 750$	$750 \leq t_C < 800$	$800 \leq t_C$
Architecture A	0 %	0 %	0 %	9.9 %	0 %	90.1 %
Architecture B	31.4 %	49.8 %	0 %	18.6 %	0 %	0.2 %

"What-ifs" and combined SODA/SDDA analysis

The following are sample questions that can be addressed with combined SODA and SDDA analysis for the Mars Exploration Architecture SoS:

1. How do capabilities change over time?
2. How to decide if, when, and how to update the SoS (choice of involved stakeholders, cost, improvement, risk)?

I used combined SODA and SDDA in the baseline case, where no failures or delays occur. Figure 5.19 shows the evolution of the operability of required capabilities over time in architecture A, and figure 5.20 shows the evolution of the operability of required capabilities over time in architecture B. Both architectures show two major steps, corresponding to the completion of the cargo transportation vehicles with the first launches, and to the arrival of crew on Mars and assemblage of the required facilities. However, architecture A reaches the first step after development and completion of the heavy launchers, that transport cargo and crew together in subsequent launches. In architecture B, various technologies are developed in parallel, reaching some plateau when operations are moved from the Earth to the Moon. The cargo is sent separately, resulting in partial achievement of capabilities on Mars. When the crew arrives on Mars, the second major step occurs. This behavior confirms the previous analysis, showing that architecture B requires more technological improvements, thus slightly decreasing the robustness, but it also shows intermediate capabilities which improve the flexibility of the architecture and react better to delays. This simple analysis can be expanded with the introduction of deterministic or stochastic failures and delays, stakeholder policies (modeled the same way as I did for launch windows, i.e. causing not just delays but entire shifts in the development of some system), and considerations about cost. Finally, the results of SODA and SDDA high- and medium-level analysis provide constraints and requirements for detailed systems engineering at the lower level.

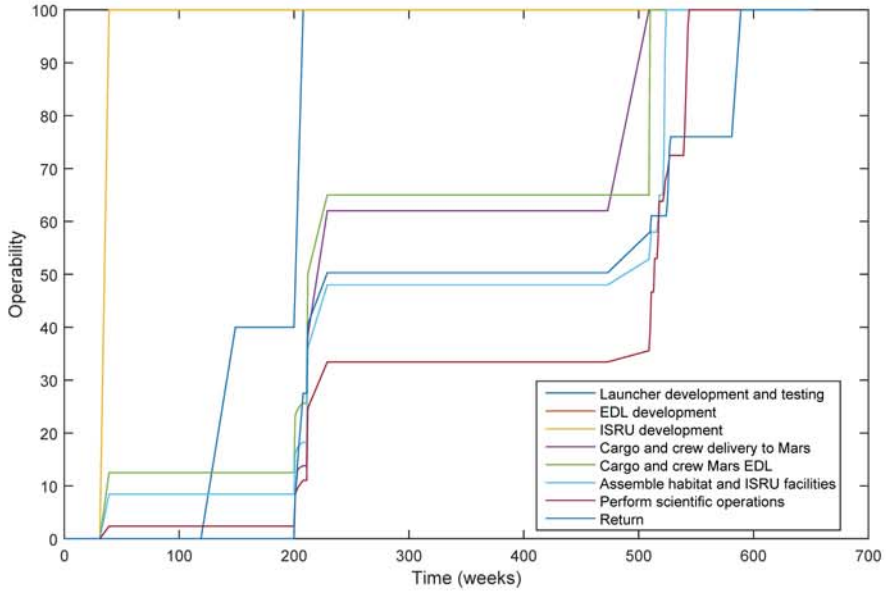


Figure 5.19. Combined SODA and SDDA analysis to quantify the evolution of capabilities over time in architecture A.

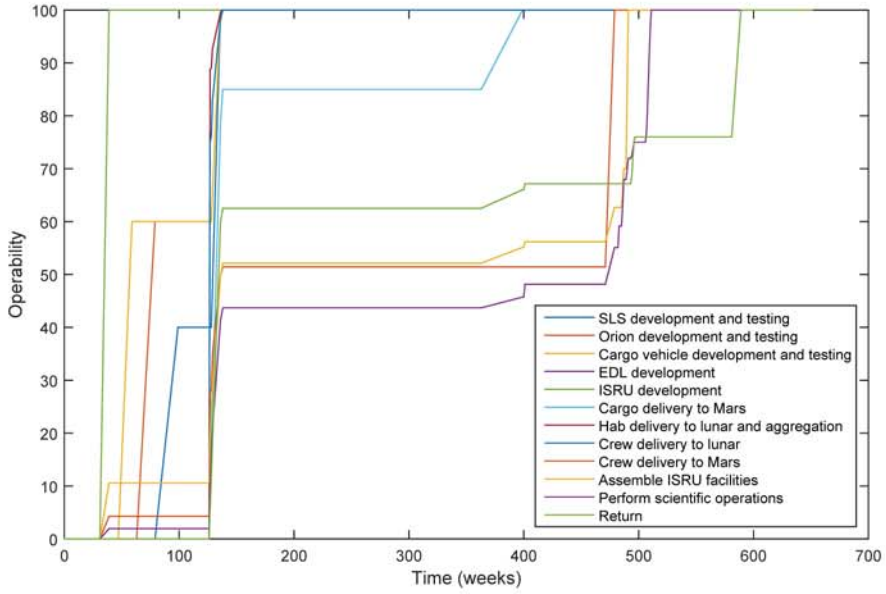


Figure 5.20. Combined SODA and SDDA analysis to quantify the evolution of capabilities over time in architecture B.

6. DISCUSSION AND CONCLUSIONS

"Do not worry about what anybody else is going to do. The best way to predict the future is to invent it."

Alan Kay,
from a meeting at Palo Alto Research Center, 1971

When determining the directions of this research, I started from five considerations:

- My passion for space exploration and space systems engineering.
- My wish to address widespread challenges and open-ended problems that pertain to various fields and areas of expertise, to put into use knowledge coming from different sources.
- The complexity and size of future endeavours in space exploration, and the sometimes hidden importance that research in space has on everyday life and will have on the future of Humanity.
- The limitation of some of the current approaches to space systems engineering, especially when combined with new needs and reduced budget.
- The need for novel methodologies to address some of the limitations of current approach without replacing traditional systems engineering methods, but supporting and expanding their usefulness.

Based on these reasons, I chose to keep in mind the problem of evaluation of architectures for Mars exploration as a guideline, but not to address this specific

problem in detail. Instead, I identified gaps and needs of complex space systems, and their commonalities with complex problems in other fields, and I developed domain-independent methodologies that could address these gaps and needs. The main categories of gaps and needs I described are

- The need for novel methods to address complexity and size in space systems. Even taking into account concurrent engineering, design processes are still monolithic and stovepiped. A holistic, high-level view, with integration and analysis of the effect of the topology of systems architecture on the overall behavior, is not yet considered, or it is limited to the end of the design process.

- The need to explicitly consider and evaluate (possibility quantitatively) the impact of dependencies on desired features. These dependencies should consider impacts and effects that go beyond a binary variable, but should be modeled accounting for possible partial dependencies.

- The need for methodology to evaluate the cascading effect of partial failures in a complex architecture, to improve reliability and risk analysis.

- The need to model development and schedule of large complex systems, accounting for dependencies in the developmental domain. This model should include parallel tasks and intelligent scheduling and re-scheduling, to quantify the impact of delays, including those due to uncertainties, Technology Readiness Level, stakeholders decisions.

Based on my considerations and on the identified gaps, I developed the Systems Operational Dependency Analysis (SODA) methodology, and the Systems Developmental Dependency Analysis (SDDA) methodology.

SODA and SDDA, addressing the operational and developmental domain respectively, are based on similar ideas. They are both parametric models of systems behavior. While they lose some detail, in comparison with other modeling tools (Surface Response Methodology, for example), SODA and SDDA trade off this lower level of

detail for various advantages. First of all, the modeling parameters have an intuitive meaning, making it easier for a user to understand the observed outcomes and to make informed decisions to improve the behavior of the systems. Second, both methods use models of one-to-one dependencies to evaluate complex cascading impact of multiple failures and delays. Third, both methods are computationally very fast, therefore they allow for simulations of thousands of instances of thousands of systems and subsystems in a reasonable time (200 time steps of 10000 instances of the 5300 systems and subsystems in the on-orbit satellite servicing SoS application required about 4 hours in Matlab environment with an Intel i3-2350 processor).

SODA and SDDA require some effort in abstraction, because their domain independence necessitates the conversion of physical measures of internal status and performance of a system to abstract levels of *operability* and *self-effectiveness*. This abstraction, however, allows for comparison of different architectures and systems, besides making SODA and SDDA applicable to a variety of problems in different fields. It is important to underline that the confidence of the user in the results of analysis with SODA and SDDA can grow higher if subject matter experts are involved in the modeling effort, and when a specific problem is studied by a diverse group, including experts in the various disciplines involved. The development of SODA and SDDA and the applications presented in this dissertation highlighted the importance of choosing levels of abstraction and dependency networks suitable for the specific problem and for the desired analysis, and of developing appropriate mapping of performance to key parameters of SODA and SDDA models. These tasks can sometimes be challenging, because they require good knowledge of the disciplines of interest in the problem under analysis, as well as a thorough understanding of SODA and SDDA methodology. Chapter 2 and sections 3.1.4 and 3.2.2 provide tips and advice for a fruitful application of SODA and SDDA.

I demonstrated the usefulness of SODA and SDDA to the aerospace and space systems engineering community with a series of sample applications. Since the goal was not to solve each individual problem, but to test and prove the methodology,

I made use of simplifying assumptions. However, these assumptions can be easily relaxed, and traditional methods can be added to SODA and SDDA analysis, to define more detailed and realistic versions of the problems.

Notwithstanding the assumptions, however, I showed how SODA can be used to model complex systems, possibly hierarchical, and to determine the most critical systems and the impact of partial and total failures on the overall behavior, identifying the weaknesses of a given architecture. While raw results of SODA analysis already give information about the features and behavior of the systems, I also proposed a metric for robustness of an architecture (capability to withstand failures and reduce their impact) and a metric for resilience of an architecture (capability to react and reshape the architecture to reduce the impact of failures). Both metrics are based on the quantification of operabilities of interest compared to the largest failure in the architecture. These metrics are useful to assess and compare different architectures, and give quantities that can be used to trade off against other quantitative measures of interest (cost, schedule, developmental risks, etc.).

I then showed how SDDA can be used to model the development and deployment of complex systems, and to determine the most critical systems on development time of the overall architecture, and the impact of delays in individual or multiple systems. I also showed how SDDA can model partial or total delay absorption due to possible parallel development of loosely dependent systems. Finally, I depicted how SDDA can be used to evaluate possible outcomes of managerial decisions, given different levels of uncertainty and possible errors in the initial assumptions for development delays.

With the last small application, I showed how SODA and SDDA can be used together, to evaluate the achievement of partial and total capabilities over time, during the development of a complex systems. Besides the immediate information about partial capabilities, these results also give information about the possible flexibility of architectures in case of stakeholder decisions, and about the impact of delays on the operability of the complex system during the development.

After the sample applications, I addressed the problem of the evaluation of architectures for Mars exploration. I framed the problem in an existing System-of-Systems three-phase analytical framework, and I showed how SODA and SDDA support these phases, giving useful results and information for high-level evaluation and decision, which add more objective and quantitative constraints and requirements to the remainder of the design process. Some of the results about robustness, impacts of delay and of failures, were unexpected and gave more insight into the advantages and disadvantages of the architectures I evaluated. Once again, the simplifying assumptions can be relaxed, and SODA and SDDA analysis can address more detailed abstraction levels of the architectures, and be used together with methods in the traditional systems design process, in an iterative fashion.

In conclusion, SODA and SDDA can give a substantial contribution to complex systems- and SoS-related problems, since they add high-level considerations and quantitative analysis of the impact of dependencies, evaluation of the cascading effect of failures and delays, information and assessment to compare architectures and support decisions in early phases of the design process.

Due to the domain-independent nature of SODA and SDDA, communities in many different fields can benefit from the use of these methodologies. For the same reason, many directions of research remain open for the future:

1. Some of the assumptions used in this dissertation can be simplified, and SODA and SDDA applied to real-world, detailed problems, in conjunction with existing methodologies

2. Expanding on what I presented in this dissertation, the research community can develop several variants of SODA and SDDA, for application to specific fields, as well as interfaces to connect SODA and SDDA to other tools

3. Researchers can propose and develop various metrics based on the raw outcomes of SODA and SDDA analysis. These metrics can help reducing the level of qualitative and expert-based evaluation of complex systems

4. SODA and SDDA can be used to support an increase in automation of the design and evaluation process

REFERENCES

REFERENCES

[1] Eugene Kranz. *Failure Is Not an Option.* Simon & Schuster Paperbacks, 2000.

[2] Jim Lovell and Jeffrey Kluger. *Apollo 13.* Coronet Books, 1994.

[3] Mark W. Maier. Architecting Principles for Systems-of-Systems. *Systems Engineering*, 1(4):267–284, 1998.

[4] Andrew P. Sage and Christopher D. Cuppan. On the systems engineering and management of systems of systems and federations of systems. *Information Knowledge Systems Management*, 2(4):325–345, 2001.

[5] Daniel DeLaurentis and Robert K. Callaway. A system-of-systems perspective for public policy decisions. *Review of Policy Research*, 21(6):829–837, 2004.

[6] Muharrem Mane, Daniel DeLaurentis, and Arthur Frazho. A Markov Perspective on Development Interdependencies in Networks of Systems. *Journal of Mechanical Design*, 133(10), 2011.

[7] Igor Nai Fovino and Marcelo Masera. Emergent disservices in interdependent systems and system-of-systems. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 590–595, 2006.

[8] John C. Hsu. Emergent behavior of systems-of-systems. In *INCOSE Mini-Conference*, Los Angeles, California, 7 February 2009.

[9] M. E. J. Newman. Resource letter cs-1: Complex systems. *American Journal of Physics*, 2011.

[10] Peter Fortescue, Graham Swinerd, and John Stark, editors. *Spacecraft systems engineering.* John Wiley & Sons, 4^{th} edition, 2011.

[11] C. Keating, R. Rogers, R. Unal, D. Dryer, A. Sousa-Poza, R. Safford, W. Peterson, and G. Rabadi. System of systems engineering. *Engineering Management Journal*, 15(3):36–45, 2003.

[12] Judith Dahmann and Kathy Smith. Integrating Systems Engineering and Test & Evaluation in System of Systems Development. In *IEEE International System Conference*, 2012.

[13] Donna H Rhodes, Adam M Ross, and Deborah J Nightingale. Architecting the System of Systems Enterprise: Enabling Constructs and Methods from the Field of Engineering Systems. In *IEEE Systems Conference*, pages 23–26, 2009.

[14] Olivier L. de Weck, Adam M. Ross, and Donna H. Rhodes. Investigating relationships and semantic sets amongst system lifecycle properties (ilities). In 3^{rd} *International Engineering Systems Symposium*, 2012.

[15] Benjamin S. Blanchard and Wolter J. Fabrycky. *Systems Engineering and Analysis*. Prentice-Hall, Inc., New Jersey, 3^{rd} edition, 1998.

[16] John E Gibson, William T Scherer, and William F Gibson. *How to do systems analysis*. John Wiley & Sons, 2007.

[17] Eberhardt Rechtin and Mark W. Maier. *The art of systems architecting*. CRC Press, 2010.

[18] Vadim Kotov. *Systems of systems as communicating structures*. Hewlett Packard Laboratories, 1997.

[19] Hans Polzer. Perspectives on interoperability, 2005.

[20] Restructuring of the strategic defense initiative (sdi) program. In *Joint hearing before the Committee on Armed Service of the United States Senate and the Committee on Armed Service of the House of Representatives, one hundredth Congress, second session*, Washington, DC, 1989. United States Government Printing Office.

[21] Mark W. Maier. Architecting principles for systems-of-systems. In *INCOSE International Symposium*, volume 6, pages 565–573, 1996.

[22] Oleg V. Sindiy. *Model-based System-of-Systems Engineering for Space-Based Command, Control, Communication, and Information Architecture Design*. PhD thesis, Purdue University, 2010.

[23] Lee W. Wagenhals and Alexander H. Levis. Service oriented architectures, the DoD architecture framework 1.5, and executable architectures. *Systems Engineering*, 12(4):312–343, 2009.

[24] Kevin H. Bonanne. A model-based approach to System-of-Systems engineering via the systems modeling language. Master's thesis, Purdue University, 2014.

[25] J. Clark Beesemyer, Adam M. Ross, and Donna H. Rhodes. An empirical investigation of system changes to frame links between design decisions and ilities. *Procedia Computer Science*, 8:31–38, 2012.

[26] Payuna Uday and Karen Marais. Designing Resilient Systems-of-Systems: A Survey of Metrics, Methods, and Challenges. *Systems Engineering*, 18(5):491–510, 2015.

[27] Oleg V. Sindiy. A system-of-systems framework for improved decision support in space exploration. Master's thesis, Purdue University, 2007.

[28] Oleg V Sindiy, Daniel A DeLaurentis, and William B Stein. An Agent-Based Dynamic Model for Analysis of Distributed Space Exploration Architectures. *Journal of the Astronautical Sciences*, 57(3):579–606, 2009.

[29] Oleg V. Sindiy, Kristopher L. Ezra, Daniel A. DeLaurentis, Barrett S. Caldwell, Thomas I. McVittie, and Kimberly A. Simpson. Analogs Supporting Design of Lunar Command, Control, Communication, and Information Architectures. *Journal of Aerospace Computing, Information, and Communication*, 7(5):151–176, 2010.

[30] Muharrem Mane and Daniel DeLaurentis. System Development and Risk Propagation in Systems-of-System. In *7th Annual Acquisition Research Symposium*, 2010.

[31] Muharrem Mane and Daniel A. DeLaurentis. Network-level metric measuring delay propagation in networks of interdependent systems. In *IEEE International Conference on System of Systems Engineering*, 2010.

[32] Muharrem Mane, Daniel Delaurentis, and Arthur Frazho. A Markov perspective on system-of-systems complexity. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1238–1243, 2011.

[33] Kartavya Neema, Shashank Tamaskar, and Daniel Delaurentis. Innovative Framework for Orbital Debris Mitigation through Satellite Rejuvenation. In *AIAA Space Conference and Exposition*, 2012.

[34] Seung Yeob Han, Karen Marais, and Daniel DeLaurentis. Evaluating system of systems resilience using interdependency analysis and competing risk model. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1251–1256, 2012.

[35] Zhemei Fang and Daniel DeLaurentis. Dynamic planning of system of systems architecture evolution. *Procedia Computer Science*, 28:449–456, 2014.

[36] Zhemei Fang and Daniel DeLaurentis. Multi-stakeholder dynamic planning of System of Systems development and evolution. *Procedia Computer Science*, 44:95–104, 2015.

[37] Navindran Davendralingam, Daniel DeLaurentis, Zhemei Fang, Cesare Guariniello, Seung Yeob Han, Karen Marais, Ankur Mour, and Payuna Uday. An analytic workbench perspective to evolution of system of systems architectures. *Procedia Computer Science*, 28:702–710, 2014.

[38] Navindran Davendralingam and Daniel DeLaurentis. A robust portfolio optimization approach to system of system architectures. *Systems Engineering*, 18(3):269–283, 2015.

[39] George E. P. Box and K.B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):1–45, 1951.

[40] André I. Khuri and Siuli Mukhopadhyay. Response surface methodology. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(2):128–149, 2010.

[41] Genichi Taguchi. *Introduction to quality engineering: designing quality into products and processes*. 1986.

[42] Judea Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, Palo Alto, CA, 1988.

[43] Marie-Lise Moullec, Marc Bouissou, Marija Jankovic, Jean-Claude Bocquet, François Réquillard, Olivier Maas, and Olivier Forgeot. Toward System Architecture Generation and Performances Assessment Under Uncertainty Using Bayesian Networks. *Journal of Mechanical Design*, 135(4), 2013.

[44] Graeme B. Shaw, David W. Miller, and Daniel E. Hastings. Development of the quantitative generalized information network analysis (gina) methodology for satellite systems. *Journal of Spacecraft and Rockets*, 38(2):257–269, 2001.

[45] T. R. Browning. Applying the Design Structure Matrix to System Decomposition and Integration Problems: a Review and New Directions. *IEEE Transactions on Engineering Management*, 48(3):292–306, 2001.

[46] Steven D Eppinger and Tyson R Browning. *Design Structure Matrix Methods and Applications*. MIT Press, 2012.

[47] Maik Maurer and Udo Lindemann. The Application of the Multiple-Domain Matrix. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 2487–2493, 2008.

[48] Jason E. Bartolomei, Daniel E. Hastings, Richard de Neufville, and Donna H. Rhodes. Engineering Systems Multiple-Domain Matrix: An Organizing Framework For Modeling Large-Scale Complex Systems. *Systems Engineering*, 15:41–61, 2012.

[49] Steven M. Rinaldi, James P. Peerenboom, and Terrence K. Kelly. Identifying, understanding, and analyzing critical infrastructure interdependencies. *IEEE Control Systems Magazine*, 21(6):11–25, 2001.

[50] Manuel E. Sosa. A structured approach to predicting and managing technical interactions in software development. *Research in Engineering Design*, 19(1):47–70, 2008.

[51] Enrico Zio and Giovanni Sansavini. Modeling Interdependent Network Systems for Identifying Cascade-Safe Operating Margins. *IEEE Transactions on Reliability*, 60(1):94–101, 2011.

[52] Dung T. Nguyen, Yilin Shen, and My T. Thai. Detecting critical nodes in interdependent power networks for vulnerability assessment. *IEEE Transactions on Smart Grid*, 4(1):151–159, 2013.

[53] Roshan S Gaonkar and N Viswanadham. Analytical fremework for the management of risk in supply chains. *IEEE Transactions on Automation Science and Engineering*, 4(2):265–273, 2007.

[54] Chao Fang and Franck Marle. A simulation-based risk network model for decision support in project risk management. *Decision Support Systems*, 52(3):635–644, 2012.

[55] Tolga Kurtoglu, Irem Y. Tumer, and David C. Jensen. A functional failure reasoning methodology for evaluation of conceptual system architectures. *Research in Engineering Design*, 21(4):209–234, 2010.

[56] H.A. Watson. Launch control safety study. Technical report, Bell labs, Murray Hill, NJ, 1961.

[57] A. B. Mearns. Fault tree analysis - the study of unlikely events in complex systems (fault tree analysis as tool to identify component failure as probable cause of undesired event in complex system). In *System Safety Symposium*, Seattle, WA, 1965.

[58] United States Department of Defense. *MIL-P-1629. Procedures for performing a failure mode effect and critical analysis*, 1949.

[59] Wendai Wang Wendai Wang, J. Loman, and P. Vassiliou. Reliability importance of components in a complex system. *Annual Symposium on Reliability and Maintainability*, pages 6–11, 2004.

[60] Hugh L. McManus, Daniel E. Hastings, and Joyce M. Warmkessel. New Methods for Rapid Architecture Selection and Conceptual Design. *Journal of Spacecraft and Rockets*, 41(1):10–19, 2004.

[61] Marco Nunez, Varun Chander Datta, Arturo Molina-Cristobal, Marin Guenov, and Atif Riaz. Enabling exploration in the conceptual design and optimisation of complex systems. *Journal of Engineering Design*, 23(10-11):849–872, 2012.

[62] Bahram Hamraz, Nicholas H. M. Caldwell, and P. John Clarkson. A Multidomain Engineering Change Propagation Model to Support Uncertainty Reduction and Risk Management in Design. *Journal of Mechanical Design*, 134(10), 2012.

[63] Manu Augustine, Om Prakash Yadav, Rakesh Jain, and Ajay Rathore. Cognitive map-based system modeling for identifying interaction failure modes. *Research in Engineering Design*, 23(2):105–124, 2012.

[64] P John Clarkson, Caroline Simons, and Claudia Eckert. Predicting Change Propagation in Complex Design. *Journal of Mechanical Design*, 126:788–797, 2004.

[65] Zhenyu Yan and Yacov Y. Haimes. Risk-based multiobjective resource allocation in hierarchical systems with multiple decisionmakers. part I: Theory and methodology. *Systems Engineering*, 14(1):1–16, 2011.

[66] Zhenyu Yan and Yacov Y Haimes. Risk-based multiobjective resource allocation in hierarchical systems with multiple decisionmakers. part II. a case study. *Systems Engineering*, 14(1):17–28, 2011.

[67] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar. Application of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669, 1959.

[68] James E Kelley and Morgan R Walker. Critical-path planning and scheduling. In *Eastern Joint IRE-AIEE-ACM Computer Conference*, volume 32, pages 160–173, 1959.

[69] D. R. Fulkerson. Expected critical path lengths in PERT networks. Technical report, the RAND corporation, 1962.

[70] Pierre Robillard and Michel Trahan. Expected completion time in PERT networks. *Operations Research*, 24(1):177–182, 1976.

[71] Esma Nur Cinicioglu and Prakash P. Shenoy. Solving stochastic PERT networks exactly using hybrid bayesian networks. In 7^{th} *Workshop on Uncertainty Processing*, pages 183–197, 2006.

[72] Amir Azaron and Reza Tavakkoli-Moghaddam. Multi-objective time-cost trade-off in dynamic PERT networks using an interactive approach. *European Journal of Operational Research*, 180(3):1186–1200, 2007.

[73] S.C. Sharma. *Operation research: Pert, Cpm & cost analysis*. Discovery Publishing House, New Delhi, India, 2006.

[74] Bennet Lientz and Kathryn Rea. *Project management for the 21st century*. Routledge, 3^{rd} edition, 2002.

[75] Donald E. Brown, John A. Marin, and William T. Scherer. A survey of intelligent scheduling systems. In Donald E. Brown and William T. Scherer, editors, *Intelligent Scheduling Systems*, chapter 1. Springer, 1995.

[76] Stephen F. Smith. Reactive scheduling systems. In *Intelligent scheduling systems*, chapter 7. Springer, 1995.

[77] Jigish S. Zaveri and Ali F. Emdad. Intelligent scheduling systems: an artificial-intelligence-based approach. In *Manufacturing Decision Support Systems*, chapter 10. Springer, 1997.

[78] Rodolfo Ambriz. *Dynamic Scheduling with Microsoft Office Project 2007: The Book by and for Professionals*. J. Ross Publishing, Inc., 2008.

[79] Barry Boehm, Jo Ann Lane, Supannika Koolmanojwong, and Richard Turner. *The incremental commitment spiral model: Principles and practices for successful systems and software*. Addison-Wesley Professional, 2014.

[80] Jerome D. Wiest and Ferdinand K. Levy. A management guide to pert/cpm, 1977.

[81] Richard Turner. A lean approach to scheduling systems engineering resources. Technical report, Defense Technical Information Center, 2013.

[82] Viswanathan Krishnan, Steven D. Eppinger, and Daniel E. Whitney. A Model-Based Framework to Overlap Product Development Activities. *Management Science*, 43(4):437–451, 1997.

[83] Mitchell Fleischer and Jeffrey K Liker. *Concurrent engineering effectiveness: integrating product development across organizations*. Hanser Gardner, Cincinnati, OH, 1997.

[84] Donald Birchler, Gary Christle, and Eric Groo. Investigating concurrency in weapons programs. Technical report, Defense Technical Information Center, 2010.

[85] Debra Facktor Lepore, John M Colombi, Jennifer Ford, Ryan Colburn, and Yosef Morris. Observations on expedited systems engineering practices in military rapid development projects. *Procedia Computer Science*, 16:502–511, 2013.

[86] Jarret M Lafleur. *A markovian state-space framework for integrating flexibility into space system design decisions*. PhD thesis, Georgia Institute of Technology, 2011.

[87] Jarret M Lafleur and John F Connolly. Integrating Flexibility into Human Space Exploration Architecture Design Decisions. In *AIAA Space Conference and Exposition*, 2012.

[88] Adrian R. L. Tatnall, John B. Farrow, Massimo Bandecchi, and C. Richard Francis. Spacecraft system engineering. In Peter Fortescue, Graham Swinerd, and John Stark, editors, *Spacecraft systems engineering*, chapter 19. John Wiley & Sons, 4^{th} edition, 2011.

[89] Systems Engineering Handbook. Technical report, NASA, 2007.

[90] James Richard Wertz, David F. Everett, and Jeffery John Puschell, editors. *Space Mission Engineering: The New SMAD*. Microcosm Press, 2011.

[91] Dale S. Caffall and James Bret Michael. Space applications of system of systems. In Mo Jamshidi, editor, *System of Systems Engineering - Principles and Applications*, chapter 15. CRC Press, Boca Raton, FL, 2009.

[92] Steven D Jolly and Brian K Muirhead. System of systems engineering in space exploration. In Mo Jamshidi, editor, *System of systems engineering, innovations for the 21th century*, chapter 14. John Wiley & Sons, 2009.

[93] Roshanak Nilchiani. *Measuring space systems flexibility: a comprehensive six-element framework*. PhD thesis, Massachusetts Institute of Technology, 2005.

[94] Charles E. Bloomquist. Spacecraft Anomalies and Lifetimes. In *Annual Reliability and Maintainability Symposium*, pages 186–191, 1984.

[95] Joel S. Greenberg. Satellite reliability requirements: effect of transportation system, sparing and maintenance strategies. *Acta Astronautica*, 21(6-7):357–365, 1990.

[96] Jean-Francois Castet and Joseph H. Saleh. Satellite reliability: statistical data analysis and modeling. *Journal of Spacecraft and Rockets*, 46(5):1065–1076, 2009.

[97] Jean-Francois Castet and Joseph H. Saleh. Interdependent Multi-Layer Networks: Modeling and Survivability Analysis with Applications to Space-Based Networks. *PLoS one*, 8(4), 2013.

[98] T. Owens Walker, Murali Tummala, and John McEachen. A system of Systems study of space-based networks utilizing picosatellite formations. In *5th International Conference on System of Systems Engineering*, 2010.

[99] Paul Garvey and Ariel Pinto. Introduction to functional dependency network analysis. In *Second International Engineering Systems Symposium*, Cambridge, Massachusetts, 15-17 June 2009.

[100] Paul Garvey and Ariel Pinto. *Advanced Risk Analysis in Engineering Enterprise Systems*. CRC Press, 2012.

[101] Paul R Garvey, C Ariel Pinto, and Joost Reyes Santos. Modelling and measuring the operability of interdependent systems and systems of systems: advances in methods and applications. *International Journal of System of Systems Engineering*, 5(1):1–24, 2014.

[102] Wassily W Leontief. Quantitative input and output relations in the economic systems of the united states. *The review of economic statistics*, 18(3):105–125, 1936.

[103] Yacov Y Haimes and Pu Jiang. Leontief-based model of risk in complex interconnected infrastructures. *Journal of Infrastructure systems*, 7(1):1–12, 2001.

[104] Yacov Y Haimes, Barry M Horowitz, James H Lambert, Joost R Santos, Chenyang Lian, and Kenneth G Crowther. Inoperability input-output model for interdependent infrastructure sectors. i: Theory and methodology. *Journal of Infrastructure Systems*, 11(2):67–79, 2005.

[105] R. Chow, D. Braun, and D. Fry. *DAF: Discrete Agent Framework Programmer's Manual.* Purdue University, West Lafayette, IN, 2012.

[106] Ankur Mour, C. Robert Kenley, Navindran Davendralingam, and Daniel DeLaurentis. Agent-based modeling for systems of systems. In *INCOSE International Symposium*, volume 23, pages 973–987, 2013.

[107] Cesare Guariniello and Daniel DeLaurentis. Dependency analysis of system-of-systems operational and development networks. *Procedia Computer Science*, 16:265–274, 2013.

[108] INCOSE. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities.* John Wiley & Sons, 2015.

[109] Robert B. Stone and Kristin L. Wood. Development of a functional basis for design. *Journal of Mechanical design*, 122(4):359–370, 2000.

[110] Ryan S. Hutcheson, Daniel A. McAdams, Robert B. Stone, and Irem Y. Tumer. Function-based behavioral modeling. In *ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 547–558. American Society of Mechanical Engineers, 2007.

[111] Irem Y Tumer and Robert B Stone. Mapping function to failure mode during component development. *Research in Engineering Design*, 14(1):25–33, 2003.

[112] Robert B. Stone, Irem Y. Tumer, and Michael Van Wie. The function-failure design method. *Journal of Mechanical Design*, 127(3):397–407, 2005.

[113] Ronald Fisher. The arrangement of field experiments. *Journal of the Ministry of Agriculture of Great Britain*, 33:503–513, 1926.

[114] Douglas C Montgomery, George C Runger, and Norma F Hubele. *Engineering statistics.* John Wiley & Sons, 2009.

[115] George E. P. Box, J. Stuart Hunter, and William Gordon Hunter. *Statistics for experimenters: design, innovation, and discovery.* Wiley-Interscience, 2 edition, 2005.

[116] Henry Laurence Gantt. *Work, wages, and profits: their influence on the cost of living.* Engineering magazine, 1910.

[117] Paul Dickson. *Sputnik: The shock of the century.* Bloomsbury Publishing, 2001.

[118] S. Tamaskar, K. Neema, T. Kotegawa, and D. DeLaurentis. Complexity enabled design space exploration. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1250–1255, 2011.

[119] Andrew Long, Matthew Richards, and Daniel E Hastings. On-orbit servicing: a new value proposition for satellite design and operation. *Journal of Spacecraft and Rockets*, 44(4):964–976, 2007.

[120] Kazuya Yoshida. Engineering test satellite vii flight experiments for space robot dynamics and control: theories on laboratory test beds ten years ago, now in orbit. *The International Journal of Robotics Research*, 22(5):321–335, 2003.

[121] Dimitar Dimitrov. *Dynamics and control of space manipulators during a satellite capturing operation.* PhD thesis, Tohoku University, 2005.

[122] Kazuya Yoshida, Dimitar Dimitrov, and Hiroki Nakanishi. On the capture of tumbling satellite by a space robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4127–4132, 2006.

[123] Cesare Guariniello, Luigi Ansalone, and Fabio Curti. Autonomous capture of non-cooperative spacecraft with a space free-flyer. In 3^{rd} *CEAS Air&Space Conference and* 21^{st} *AIDAA Congress*, 2011.

[124] On-orbit satellite servicing study. Technical report, NASA, 2010.

[125] Brook Rowland Sullivan. *Technical and economic feasibility of telerobotic on-orbit satellite servicing.* PhD thesis, University of Maryland, 2005.

[126] Bruno Ciciani and Vincenzo Grassi. Performability evaluation of fault-tolerant satellite systems. *IEEE Transactions on Communications*, 35(4):403–409, 1987.

[127] Vincenzo Grassi, Lorenzo Donatiello, and Giuseppe Iazeolla. Performability evaluation of multicomponent fault-tolerant systems. *IEEE Transactions on Reliability*, 37(2):216–222, 1988.

[128] Mak Tafazoli. A study of on-orbit spacecraft failures. *Acta Astronautica*, 64(2):195–205, 2009.

[129] Brook R Sullivan and David L Akin. A survey of serviceable spacecraft failures. In *AIAA Space Conference and Exposition*, 2001.

[130] John E Prussing and J-H Chiu. Optimal multiple-impulse time-fixed rendezvous between circular orbits. *Journal of Guidance, Control, and Dynamics*, 9(1):17–22, 1986.

[131] B. P. Abbott. Littoal Combat Ship (LCS) Mission Packages: determining the best mix. Master's thesis, Naval Postgraduate School, 2008.

[132] Homayoon Dezfuli. Achieving a holistic and risk-informed decision-making process at nasa. In *Workshop on Tolerable Risk Evaluation*, Arlington, VA, 2008.

[133] Homayoon Dezfuli. Overview of the nasa's system safety approach in the context of risk-informed decision making. In *Trilateral Safety and Mission Assurance Conference*, Noordwijk, The Netherlands, 2008.

[134] Daniel DeLaurentis. Understanding transportation as a system-of-systems design problem. In 43^{rd} *AIAA Aerospace Sciences Meeting and Exhibit*, 2005.

[135] Bret G Drake, Stephen J Hoffman, and David W Beaty. Human exploration of mars, design reference architecture 5.0. In *IEEE Aerospace Conference*, 2010.

[136] K. Klaus, M. L. Raftery, and K. E. Post. An affordable mars mission design. In 45^{th} *Lunar and Planetary Science Conference*, 2014.

APPENDICES

A. BETA PROBABILITY DENSITY FUNCTION

The Beta distribution is a family of continuous probability distributions defined on the interval $[0, 1]$, parameterized by two positive shape parameters, denoted by α and β, that appear as exponents of the random variable and control the shape of the distribution.

The probability density function of the random variable x is defined as

$$\frac{x^{\alpha-1}\left(1-x\right)^{\beta-1}}{B\left(\alpha,\beta\right)}$$

where B is related to the Γ distribution, with $B\left(\alpha,\beta\right)=\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$.

The expected value of the random variable is $E\left[X\right]=\frac{\alpha}{\alpha+\beta}$, and the variance is $\sigma^2\left[X\right]=\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$.

The Beta distribution is used in a variety of disciplines, including genetics, project management, stratigraphy, and biology. It is often used to model the initial knowledge about the probability of a space vehicle to successfully complete a specified mission.

In this dissertation, to generate values of self-effectiveness, I multiply the instances of the random variable x, generated out of Beta distributions, by a factor of 100, so that the interval ranges between 0 and 100. The variety of the distribution families based on the shape parameters (figure A.1), the finite range of the random variable, and the simple expression of the expected value and variance of the random variable are the reasons why I used Beta distribution in this research.

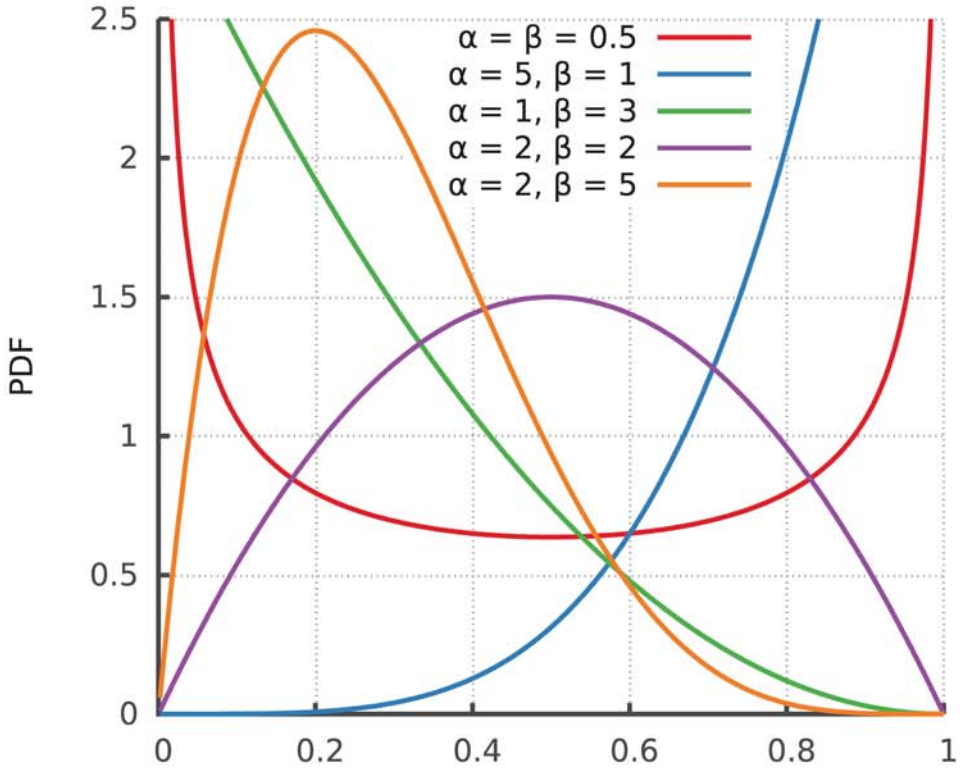


Figure A.1. Probability density function of the Beta distribution family. Source: Wikipedia.

B. MARS EXPLORATION ARCHITECTURE - β LEVEL

This appendix lists the component systems and the matrices of SDDA and SODA parameters of the two architectures for Mars exploration analyzed in chapter 5.

B.1 Architecture A

B.1.1 SDDA

Table B.1. Systems and capabilities in the developmental network of architecture A

1	Launch sites
2	Control centers
3	International Space Station
4	Heavy launcher 1
5	Heavy launcher 2
6	Heavy launcher 3
7	Mars habitat
8	In-situ resources facility
9	In-situ mobility vehicle
10	Crew return vehicle
11	Launcher development and testing
12	EDL development
13	ISRU development
14	Cargo and crew delivery to Mars
15	Cargo and crew Mars EDL
16	Assemble habitat and ISRU facilities
17	Perform scientific operations
18	Return
19	Re-entry

Table B.2. developmental SOD of architecture A

$$
\text{SOD} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0.6 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0.4 & 0.5 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 1 & 0 & 0 & 0 \\
0.3 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0.3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.7 & 0.8 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0.7 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Table B.3. developmental COD of architecture A

$$
\text{COD} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 & 0 & 45 & 0 & 0 & 0 & 0 & 0 & 0 \\
40 & 45 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\
30 & 25 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 70 & 0 & 100 & 0 & 0 & 0 \\
20 & 15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 & 100 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 55 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 20 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 60 & 75 & 85 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 0 & 0 & 55 & 55 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 99 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

B.1.2 SODA

Table B.4. Systems and capabilities in the operational network of architecture A

1	Launch sites
2	Control centers
3	Mars communication network
4	Heavy launcher 1
5	Heavy launcher 2
6	Heavy launcher 3
7	Mars habitat
8	In-situ resources facility
9	In-situ mobility vehicle
10	Crew return vehicle
11	Launcher development and testing
12	EDL development
13	ISRU development
14	Cargo and crew delivery to Mars
15	Cargo and crew Mars EDL
16	Assemble habitat and ISRU facilities
17	Perform scientific operations
18	Return
19	Re-entry

Table B.5. operational SOD of architecture A

$$
\text{SOD} = \begin{bmatrix}
0 & 0 & 0 & 0.6 & 0.6 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0.3 & 0.2 & 0.1 & 0.8 & 0.6 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0.8 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.9 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.9 & 0.3 & 0.1 \\
0 & 0 & 0.6 & 0 & 0 & 0 & 0.6 & 0.6 & 0.3 & 0.7 & 0 & 0 & 0 & 0.5 & 0.7 & 0.7 & 0 & 0 & 0 \\
0 & 0 & 0.6 & 0 & 0 & 0 & 0.4 & 0.5 & 0.3 & 0.6 & 0 & 0 & 0 & 0.5 & 0.7 & 0.6 & 0 & 0 & 0 \\
0 & 0 & 0.6 & 0 & 0 & 0 & 0.2 & 0.4 & 0.3 & 0.5 & 0 & 0 & 0 & 0.5 & 0.7 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.9 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.85 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0.9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 \\
0 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0 & 0.9 & 0.6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0.6 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Table B.6. operational COD of architecture A

$$
\mathrm{COD} = \begin{bmatrix}
0 & 0 & 0 & 75 & 75 & 75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 85 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 70 & 60 & 60 & 15 & 15 & 5 & 65 & 30 & 0 & 0 & 85 & 0 & 0 & 0 & 75 & 35 \\
0 & 0 & 0 & 0 & 0 & 0 & 65 & 80 & 45 & 0 & 0 & 0 & 0 & 0 & 0 & 34 & 72 & 12 & 5 \\
0 & 0 & 45 & 0 & 0 & 0 & 50 & 40 & 30 & 45 & 0 & 0 & 0 & 65 & 55 & 35 & 0 & 0 & 0 \\
0 & 0 & 45 & 0 & 0 & 0 & 40 & 35 & 30 & 40 & 0 & 0 & 0 & 65 & 55 & 30 & 0 & 0 & 0 \\
0 & 0 & 45 & 0 & 0 & 0 & 30 & 30 & 30 & 30 & 0 & 0 & 0 & 65 & 55 & 25 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 48 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 22 & 35 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 44 & 56 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 85 & 80 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 0 & 0 & 0 & 35 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 0 & 0 & 0 \\
0 & 0 & 70 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 70 & 0 & 85 & 55 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 60 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Table B.7. operational IOD of architecture A

$$
\text{IOD} = \begin{bmatrix}
0 & 0 & 0 & 12 & 12 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 & 50 & 35 & 30 & 30 & 40 & 10 & 0 & 0 & 20 & 0 & 0 & 0 & 30 & 30 \\
0 & 0 & 0 & 0 & 0 & 0 & 20 & 80 & 15 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 15 & 50 & 20 \\
0 & 0 & 55 & 0 & 0 & 0 & 70 & 60 & 10 & 45 & 0 & 0 & 0 & 20 & 12 & 35 & 0 & 0 & 0 \\
0 & 0 & 55 & 0 & 0 & 0 & 40 & 30 & 10 & 40 & 0 & 0 & 0 & 20 & 12 & 35 & 0 & 0 & 0 \\
0 & 0 & 55 & 0 & 0 & 0 & 20 & 15 & 10 & 30 & 0 & 0 & 0 & 20 & 12 & 35 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 & 55 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 20 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 75 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 55 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\
0 & 0 & 70 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 65 & 65 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 35 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 55 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

B.2 Architecture B

B.2.1 SDDA

Table B.8. Systems and capabilities in the developmental network of architecture B

1	Launch sites
2	Control centers
3	International Space Station
4	SLS 1
5	SLS 2
6	Orion 1
7	Orion 2
8	Cargo launch vehicle
9	Moon base 1
10	Moon base 2
11	Mars habitat
12	In-situ resources facility
13	In-situ mobility vehicle
14	Crew return vehicle
15	SLS development and testing
16	Orion development and testing
17	Cargo vehicle development and testing
18	EDL development
19	ISRU development
20	Cargo delivery to Mars
21	Hab delivery to lunar and aggregation
22	Crew delivery to lunar
23	Crew delivery to Mars
24	Assemble ISRU facilities
25	Perform scientific operations
26	Return
27	Re-entry

Table B.9. developmental SOD of architecture B

$$
\text{SOD} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0.1 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.3 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.1 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.25 & 0.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0.3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.7 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.7 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Table B.10. developmental COD of architecture B

$$\text{COD} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 80 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 70 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0.1 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
20 & 18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 80 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 15 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 40 & 65 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 40 & 80 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 70 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 80 & 100 & 100 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

B.2.2 SODA

Table B.11. Systems and capabilities in the operational network of architecture B

1	Launch sites
2	Control centers
3	Mars communication network
4	SLS 1
5	SLS 2
6	Orion 1
7	Orion 2
8	Cargo launch vehicle
9	Moon base 1
10	Moon base 2
11	Mars habitat
12	In-situ resources facility
13	In-situ mobility vehicle
14	Crew return vehicle
15	SLS development and testing
16	Orion development and testing
17	Cargo vehicle development and testing
18	EDL development
19	ISRU development
20	Cargo delivery to Mars
21	Hab delivery to lunar and aggregation
22	Crew delivery to lunar
23	Crew delivery to Mars
24	Assemble ISRU facilities
25	Perform scientific operations
26	Return
27	Re-entry

Table B.12. operational SOD of architecture B

$$SOD = \begin{bmatrix}
0 & 0 & 0 & 0.6 & 0.6 & 0.4 & 0.4 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0.5 & 0.4 & 0.4 & 0.55 & 0.25 & 0.25 & 0.3 & 0.2 & 0.1 & 0.8 & 0.6 & 0.4 & 0.4 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.9 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.9 & 0.3 & 0.1 \\
0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0.45 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.8 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0.3 & 0.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.8 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.85 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.85 & 0.8 & 0.75 & 0.95 & 0 & 0 & 0 & 0 & 0 & 0.95 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0.2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.9 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.85 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0.9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0.4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0 \\
0 & 0 & 0.95 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0 & 0.9 & 0.6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.75 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Table B.13. operational COD of architecture B

$$
COD = \begin{bmatrix}
0 & 0 & 0 & 65 & 65 & 30 & 30 & 65 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 60 & 60 & 30 & 30 & 45 & 15 & 15 & 45 & 15 & 12 & 60 & 55 & 35 & 35 & 0 & 0 & 75 & 0 & 0 & 0 & 0 & 0 & 62 & 45 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 85 & 52 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 75 & 38 & 15 \\
0 & 0 & 0 & 0 & 0 & 85 & 0 & 0 & 56 & 48 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 65 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 85 & 0 & 48 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 35 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 45 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 35 & 0 & 0 & 0 & 0 \\
0 & 0 & 75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 80 & 75 & 65 & 85 & 0 & 0 & 0 & 0 & 0 & 85 & 0 & 0 & 0 & 65 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 20 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 22 & 0 & 15 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 85 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 75 & 75 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 85 & 85 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 0 & 0 & 0 & 35 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 0 & 0 & 0 \\
0 & 0 & 85 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 70 & 0 & 85 & 55 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 55 & 60 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Table B.14. operational IOD of architecture B

$$
\mathrm{IOD} = \begin{bmatrix}
0 & 0 & 0 & 25 & 20 & 50 & 50 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 & 50 & 50 & 35 & 30 & 30 & 40 & 10 & 8 & 55 & 40 & 40 & 40 & 0 & 0 & 65 & 0 & 0 & 0 & 0 & 0 & 45 & 30 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 80 & 15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 20 & 25 & 10 \\
0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 & 44 & 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 42 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 40 & 28 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 & 30 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 & 30 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 25 & 0 & 0 & 0 & 0 \\
0 & 0 & 65 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 20 & 20 & 35 & 0 & 0 & 0 & 0 & 0 & 65 & 0 & 0 & 0 & 40 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 & 0 & 10 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 8 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 80 & 70 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 65 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 70 & 70 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 0 & 0 & 0 \\
0 & 0 & 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 70 & 0 & 65 & 65 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 55 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

VITA

VITA

Born in Sant'Arsenio (SA), Italy, Cesare Guariniello grew up in Rome, Italy. In 1999, he graduated at the top of his class from the *Liceo Classico Torquato Tasso* in Rome. He subsequently enrolled in the Department of Computer Engineering at the University of Rome "*La Sapienza*", in the *Laurea* study program, equivalent to a B.S. degree plus a M.S. degree. In May 2007, he graduated *cum laude* with major in Automation and Robotics, discussing his experimental research thesis entitled *Decentralized Algorithms for Sensor-based Multi-robot Exploration*. In November 2007 he passed the State Examination to be enrolled in the Italian Association of Engineers in all the branches of engineering.

In September 2008, while serving as professor of Industrial Automation at the University of Rome "*La Sapienza*", Cesare enrolled in the School of Aerospace Engineering in the same university, in the graduate program of Astronautical Engineering. During these years, he cultivated his passion for space in a program involving studies in systems, propulsion, structures, attitude dynamics, astrodynamics, space robotics, space mission design, human missions in space, space law, and history of space exploration. In July 2011, he received a M.S. degree *cum laude*, discussing his research thesis entitled *Autonomous Rendezvous and Capture of a Non-cooperative Spacecraft with a Space Free-Flyer*, where he proposed an innovative solution to the inverse dynamics of robotic manipulators on a space free-flyer.

Having accepted enrollment in the Ph.D. program in the School of Aeronautics and Astronautics at Purdue, he was offered a Research Assistantship to conduct research in the System-of-Systems laboratory, under the leadership of Dr. Daniel DeLaurentis, and he moved to Purdue in August 2011. While completing his coursework, focused on dynamics and control and aerospace systems, Cesare began his research in System-of-Systems, working on a project funded by the United States department of Defense

through the Systems Engineering Research Center (SERC). The project is currently ongoing, and its goal is the development of theoretical formulations and software tools for the analysis of complex interdependent systems.

Since 2012, Cesare is an active member of the alpha chapter of the honor society of aerospace engineering, Sigma Gamma Tau, the American Institute of Aeronautics and Astronautics (AIAA), the Institute of Electrical and Electronical Engineers (IEEE), and the American Astronautical Society (AAS). He authored and presented several conference papers, including presentations at the Conference on Systems Engineering Research (CSER), the AIAA Space Conference, and the International Astronautical Congress (IAC). More recently, Cesare published a paper in the high-impact factor journal Research in Engineering Design and submitted a paper, currently under review, to Management Science.

In 2013, Cesare was a co-founder of the Purdue University Student Division of the International Council on Systems Engineering (INCOSE), where he served as secretary for three years. In 2016, he was presented with the Leadership Award for his effort and service to the Purdue University INCOSE Student Division. In 2014, Cesare joined the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University. He was invited to present applications of his research to cybersecurity, and in 2016 he was awarded the prestigious CERIAS Diamond Award for outstanding academic achievements.

While expanding and improving his research in System-of-Systems, through collaborations, engagement with industry, and presentations to potential users of this research (including NASA, the MITRE corporation, the United States Navy, Jacobs Engineering, and the Sandia National Laboratory), in August 2014 Cesare enrolled in the department of Earth, Atmospheric, and Planetary Science (EAPS) at Purdue University. He is conducting research in the Planetary Geology group, under the leadership of Dr. Briony Horgan, and is expected to receive a M.S. degree in the Spring of 2017, with a thesis on remote sensing for the identification of materials for in-situ

resource utilization, to support the selection of landing sites for human missions to Mars.

Outside academia, Cesare enjoys a wide variety of activities. In Italy, he volunteered as an educator for fifteen years, served as a tenor in the Choir of the Diocese of Rome for twelve years, participating to services and concerts in Europe and in the United States, and played with the Italian Volleyball Federation. He also became member of the Mensa Club and the Planetary Society. Since he joined Purdue, he became a member of Purdue Fencing Club, representing Purdue in various collegiate tournaments, recently serving as captain of the men's foil team, and receiving the Silver Mask medal for his commitment to the club. He also obtained all three levels of amateur radio license from the Federal Communications Commission (FCC), and he is a member of the American Radio Relay League (ARRL) and a volunteer examiner. In 2014 he obtained his first scuba diving certification, and in 2016 he obtained his Private Pilot License (PPL) from the Federal Aviation Administration (FAA).