

2007

Incorporating product-line engineering techniques into agent-oriented software engineering for efficiently building safety-critical, multi-agent systems

Joshua Jon Dehlinger
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/rtd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Dehlinger, Joshua Jon, "Incorporating product-line engineering techniques into agent-oriented software engineering for efficiently building safety-critical, multi-agent systems" (2007). *Retrospective Theses and Dissertations*. 15529.
<https://lib.dr.iastate.edu/rtd/15529>

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

**Incorporating product-line engineering techniques into agent-oriented
software engineering for efficiently building safety-critical, multi-agent systems**

by

Joshua Jon Dehlinger

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Robyn R. Lutz, Major Professor
Samik Basu
Carl K. Chang
Manimaran Govindarasu
Suraj C. Kothari
Gary T. Leavens

Iowa State University

Ames, Iowa

2007

Copyright © Joshua Jon Dehlinger, 2007. All rights reserved.

UMI Number: 3274890



UMI Microform 3274890

Copyright 2007 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

TABLE OF CONTENTS

LIST OF FIGURES	iv
LIST OF TABLES	vii
LIST OF ACRONYMS AND ABBREVIATIONS	viii
ACKNOWLEDGEMENTS	ix
ABSTRACT	xii
CHAPTER 1. INTRODUCTION	1
1.1 Product-Line Engineering for Agent-Based Systems	2
1.2 Safety Analysis for Safety-Critical Software Product Lines	6
1.3 Safety Analysis for Multi-Agent System Product Lines	9
1.4 Statement of Thesis	13
1.5 Outline	15
CHAPTER 2. RELATED WORK	16
2.1 Software Product-Line Engineering	16
2.2 Agent-Oriented Software Engineering	23
2.3 Software Safety Analysis	30
CHAPTER 3. CASE STUDY: THE PROSPECTING ASTEROID MISSION	36
3.1 The Autonomous Nano-Technology Swam Mission	36
3.2 The Prospecting Asteroid Mission	39
CHAPTER 4. DEVELOPING MULTI-AGENT SYSTEM PRODUCT LINES USING THE GAIA-PL METHODOLOGY	46
4.1 Integrating Software Product-Line Engineering Principles into the Gaia Methodology	47
4.2 Documenting the Requirements Specifications of a MAS-PL in the Gaia-PL Methodology	57
4.3 Requirements Specifications Reuse in the Gaia-PL Methodology	91
4.4 Evaluation of the Gaia-PL Methodology	100
4.5 Summary	108
CHAPTER 5. SAFETY ANALYSIS FOR SAFETY-CRITICAL MULTI- AGENT SYSTEM PRODUCT LINES	110
5.1 Software Safety Analysis for Multi-Agent System Product Lines	111
5.2 Software Failure Modes, Effects and Criticality Analysis for the Gaia-PL Methodology	116
5.3 Product-Line Software Fault Tree Analysis and PLFaultCAT	136
5.4 Bi-Directional Safety Analysis for Multi-Agent System Product Lines ..	198
5.5 Summary	202
CHAPTER 6. CONCLUSION	205
6.1 Support for the Thesis	205

6.2 Summary of Contributions.....	210
6.3 Future Work.....	213
6.3 Summary	216
BIBLIOGRAPHY.....	218
APPENDIX A. COMMONALITY AND VARIABILITY ANALYSIS	229
APPENDIX B. PARAMETERS OF VARIATION	244
APPENDIX C. FEATURE MODEL.....	250
APPENDIX D. GAIA-PL ROLE SCHEMAS	252
APPENDIX E. SOFTWARE FAILURE MODES, EFFECTS AND CRITICALITY ANALYSIS.....	302
APPENDIX F. PRODUCT-LINE SOFTWARE FAULT TREE ANALYSIS ..	322

LIST OF FIGURES

Figure 1	The Gaia Models and their Relationships (from [93]).....	27
Figure 2	The Family of NASA’s Proposed ANTS-Based Missions.....	37
Figure 3	PAM Spacecraft Exploring the Asteroid Belt (from [76]).....	43
Figure 4	An Overview of the Software Engineering Artifacts of Gaia-PL.....	58
Figure 5	Excerpt of the Commonalities from the Commonality and Variability Analysis for the PAM MAS-PL.....	61
Figure 6	Excerpt of the Variabilities from the Commonality and Variability Analysis for the PAM MAS-PL.....	62
Figure 7	Documenting the Commonality and Variability Requirements in DECIMAL	65
Figure 8	Documenting the A Variability Requirement and its Parameters of Variation Using DECIMAL.....	65
Figure 9	Feature Model Derived from the Commonality and Variability Analysis for the PAM MAS-PL.....	68
Figure 10	The Requirements Specifications for the Navigator Role Documented in a Role Schema.....	70
Figure 11	The Role Variation Points Schema for the Self-Optimizer Role	71
Figure 12	The Variation Points Schema for the Core Variation Point of the Self- Optimizer Role.....	73
Figure 13	The Variation Points Schema for the Leader Variation Point of the Self- Optimizer Role.....	74
Figure 14	The Variation Points Schema for the Messenger Variation Point of the Self-Optimizer Role	75
Figure 15	The Variation Points Schema for the Worker Variation Point of the Self- Optimizer Role.....	76
Figure 16	A Portion of the PAM Feature Model to Illustrate Hierarchical Role Variation Points Schemas	79
Figure 17	An Excerpt of the Role Variation Points Schema for the Self-Protector Role.....	81
Figure 18	An Excerpt of the Role Variation Points Schema for the SolarStormWarner Role	81
Figure 19	The Variation Points Schema for the Passive Variation Point of the SolarStormWarner Role.....	82
Figure 20	The Variation Points Schema for the Warm-Spare Variation Point of the SolarStormWarner Role.....	83
Figure 21	The Variation Points Schema for the Active Variation Point of the SolarStormWarner Role.....	84
Figure 22	The Variation Points Schema for the SolarStormProtector Variation Point of the Self-Protector Role	85
Figure 23	A Portion of the PAM Feature Model for the SolarStormWarner Role with only the Passive Variation Point.....	87

Figure 24	Role Deployment Schema for a Configuration of the SolarStormWarner Role	88
Figure 25	Role Deployment Schema for a Configuration of the SolarStormWarner Role	88
Figure 26	An Excerpt of the Agent Model for the PAM MAS-PL	91
Figure 27	Updated Role Variation Points Schema for the Self-Optimizer Role as a Result of Evolution	98
Figure 28	The New Variation Points Schema for the Scout Variation Point of the Self-Optimizer Role as a Result of Evolution.....	99
Figure 29	Excerpt of the Updated Agent Model to Reflect the Addition of the Scout Variation Point to the Self-Optimizer Role	100
Figure 30	An Overview of the Safety Analyses for MAS-PL in the Gaia-PL Methodology	114
Figure 31	The Variation Point Schema for the CollisionProtector Role	117
Figure 32	An Overview of the PL-SFTA Safety Analysis Technique	138
Figure 33	An Overview of DECIMAL and PLFaultCAT's Role in the Design and Development of Safety-Critical Product Lines.....	140
Figure 34	PLFaultCAT's Software Architecture.....	143
Figure 35	PL-SFTA_CREATE Algorithm	148
Figure 36	An Excerpt of an Intermediate Node Tree for the Spacecraft to Asteroid Collision Hazard	149
Figure 37	An Excerpt of an Intermediate Node Tree for the Spacecraft Received Solar Storm Damage Hazard	150
Figure 38	An Intermediate Software Fault Tree for the Spacecraft to Asteroid Collision Hazard in PLFaultCAT	153
Figure 39	An Intermediate Software Fault Tree for the Spacecraft Received Solar Storm Damage Hazard in PLFaultCAT	154
Figure 40	Depicting the Influence of a Commonality for the Spacecraft to Asteroid Collision Fault Tree in PLFaultCAT	155
Figure 41	Depicting the Influence of a Variability for the Spacecraft Received Solar Damage Fault Tree in PLFaultCAT.....	156
Figure 42	Automatically Linking a Product-Line Requirement to a Software Fault Tree Node in PLFaultCAT.....	159
Figure 43	A PL-SFTA for the Spacecraft to Asteroid Collision Hazard in PLFaultCAT.....	162
Figure 44	A PL-SFTA for the Spacecraft Received Solar Damage Hazard in PLFaultCAT.....	163
Figure 45	A Generic Product-Line Software Fault Tree Analysis	165
Figure 46	Selecting the Depth to Search for the Single-Point Failures of a PL-SFTA ..	167
Figure 47	The Single-Point Failure Report Produced by PLFaultCAT.....	167
Figure 48	Updating the PL-SFTA to Mitigate Against a Single-Point Failure	169
Figure 49	The Variability Contribution Failure Report Produced by PLFaultCAT	171
Figure 50	The Variability Selection Window to Prune a PL-SFTA	174
Figure 51	Pruning the PL-SFTA in PLFaultCAT	177
Figure 52	Pruning for a Product-Line Member Software Fault Tree in PLFaultCAT ..	179

Figure 53	Resulting Product-Line Member Software Fault Tree after Manual Pruning.....	182
Figure 54	Depicting the Influence of a Role's Variation Point for the Spacecraft Received Solar Damage Fault Tree in PLFaultCAT	186
Figure 55	Variation Point Selection to Derive an Agent's PL-SFTA in PLFaultCAT .	188

LIST OF TABLES

Table 1	Types of Specialized Instruments for <i>Worker</i> Spacecraft	41
Table 2	Excerpt of the Parameters of Variation Table for the PAM MAS-PL.....	63
Table 3	A Portion of the SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role.....	121
Table 4	A Portion of the SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role.....	122
Table 5	A Portion of the SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role.....	123
Table 6	A Portion of the SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role.....	126
Table 7	A Portion of the SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role.....	127
Table 8	A Portion of the SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role.....	128
Table 9	Results of the Application of PL-SFTA to the PAM Case Study.....	190

LIST OF ACRONYMS AND ABBREVIATIONS

ANTS	Autonomous Nano-Technology Swarm
AOSE	Agent-Oriented Software Engineering
BDSA	Bi-Directional Safety Analysis
CVA	Commonality and Variability Analysis
FAST	Family-Oriented Abstraction, Specification and Translation
FMEA	Failure Modes and Effects Analysis
Gaia-PL	Gaia – Product Line
MAS	Multi-Agent System
MAS-PL	Multi-Agent System Product Line
NASA	National Aeronautics and Space Administration
PAM	Prospecting Asteroid Mission
PL-SFTA	Product-Line Software Fault Tree Analysis
SFMEA	Software Failure Modes and Effects Analysis
SFMECA	Software Failure Modes, Effects and Criticality Analysis
SFT	Software Fault Tree
SFTA	Software Fault Tree Analysis
UML	Unified Modeling Language
XML	Extensible Markup Language

ACKNOWLEDGEMENTS

Despite my name solely being on the cover of this dissertation, a great many people have invaluable contributed to its production. I owe my gratitude to those family members, friends, classmates and professors who have been instrumental to making this dissertation successful and my graduate student experience memorable.

My deepest gratitude is to my advisor, Dr. Robyn R. Lutz. I have been incredibly fortunate to have an advisor who simultaneously gave me the freedom to explore research ideas on my own and guided me towards those problems and ideas that are important. Robyn's questions, suggestions and patience have shaped this dissertation and its research. Her sincere desire to build safer software systems has inspired much of this research, and her interest in supporting space exploration has renewed my interest in the same. I am deeply grateful to her for investing the time and energy in me as a graduate student. I can only aspire to become as effective of a researcher, professor and advisor for my future students as Robyn was for me.

I would like to thank all the members of my committee: Dr. Samik Basu, Dr. Carl K. Chang, Dr. Manimaran Govindarasu, Dr. Suraj C. Kothari and Dr. Gary T. Leavens. I consider myself lucky to have such an exceptional doctoral committee. Their questions, comments, support and encouragement greatly benefited this dissertation.

In particular, I would like to thank Dr. Gary T. Leavens for providing numerous and helpful comments on an earlier draft of this dissertation. His suggestions certainly improved the presentation and clarity of this work.

Qian Feng has provided me with the unwavering love, support and encouragement needed to survive the tumultuous times of a graduate student. This dissertation would not have been possible without her.

I am thankful for all of the past and current members of the Laboratory for Software Safety: Jing (Janet) Liu, Hongyu Sun, Wei Zhang, Kendra Schmid, Lada Suvorov, Meredith Humphrey, Dingding Lu, Oko Swai, Barbara Nsiah and Miriam Hauptman. I can't imagine a better group of people as labmates and friends. Being able to chat with them, about research or otherwise, while preventing them from doing their work certainly made the rigors of graduate studies manageable. Also, the occasional ping pong matches served as much needed breaks and an outlet for stress that comes with writing a dissertation.

In particular, I am grateful to Janet Liu who has been a close friend, labmate and collaborator throughout my time at Iowa State University. I am indebted to the time she spent reviewing my work, asking probing questions and for her thoughtful suggestions and discussions. I have thoroughly enjoyed the research we have worked on together and have learned greatly from her.

I was privileged to have worked with two wonderful undergraduate students, Meredith Humphrey and Lada Suvorov, in the redesign, development and improvement of DECIMAL. In this work, I'd also like to thank Prasanna Padmanabhan, the original developer of DECIMAL, for his helpful comments, insights and suggestions.

I enjoyed the three semesters that I spent working with Dr. Simata Mitra as a teaching assistant. In that time, I learned a lot about what it takes to design, manage and teach a software engineering class. I will carry these lessons, as well as those lessons I have learned in observing Dr. Robyn R. Lutz, Dr. Hridesh Rajan and Dr. Gary T. Leavens, with me as go forward.

I am grateful to Linda Dutton for her assistance in navigating the maze of administrative paperwork throughout my time as a graduate student. She always greeted me with a smile, knew the answers to my questions and borrowed me her keys on the several occasions that I locked myself out of our research lab.

Most importantly, none of this would have been possible without the love and support of my family. I thank my parents for instilling in me the work ethic, patience, dedication and discipline needed to do whatever I undertake well. I am grateful to my father, Dr. Jonathan Dehlinger, for taking the time to read through many of the papers I have written while a graduate student and providing helpful comments despite not fully understanding the work and always providing helpful advice. I am also grateful to my mother, Sandy Dehlinger, for always worrying about me and encouraging me. My sisters, Erin and Sara Dehlinger, have given me the glimpses into the joys of the “real world” that has inspired me to finish this dissertation so that I too can enjoy the fruits of my education.

Finally, this research was supported through National Science Foundation grants 0204139 “Safety Analysis for Critical Product Lines”, 0205588 “Natural Language in the Development of High-Confidence Software”, 0541163 “Safety Analysis of Evolving Product Lines” and by the Iowa Space Grant Consortium grant “A Product-Family Approach to the Maintenance and Evolution of Constellations”.

ABSTRACT

Safety-critical, agent-based systems are being developed without mechanisms and analysis techniques to discover, analyze and verify software requirements and prevent potential hazards. Agent-oriented, software-based approaches have provided powerful and natural high-level abstractions in which software developers can understand, model and develop complex, distributed systems. Yet, the realization of agent-oriented software development partially depends upon whether agent-based software systems can achieve reductions in development time and cost similar to other reuse-conscious software development methods. Further, agent-oriented software engineering (AOSE) currently does not adequately address: (1) requirements (specification) reuse in a way that is amenable to the reduction of the development cost by utilizing reusable assets, and (2) analysis techniques to evaluate safety.

This dissertation offers our AOSE methodology, Gaia-PL (Gaia – Product Line) for open, agent-based distributed software systems to capture requirements specifications that can be easily reused. Our methodology uses a product-line perspective to promote reuse in agent-based, software systems early in the development lifecycle so that software assets can be reused throughout the development lifecycle and system evolution.

The main contribution of this work is a requirements specification pattern that captures the dynamically changing design configurations of agents. Reuse is achieved by adopting a product-line approach into AOSE. Requirements specifications reuse is the ability to easily use previously defined requirements specifications from an earlier system and apply them to a new, slightly different system. This can significantly reduce the development time and cost of building an agent-based system.

For safety-critical agent-based systems, this dissertation incorporates reuse-oriented safety analysis methods for AOSE to allow the discovery of new safety requirements and the verification that the design satisfies the safety requirements. Specifically, Product-Line Software Fault Tree Analysis (PL-SFTA) and its automated tool, PLFaultCAT (**P**roduct-**L**ine **F**ault Tree **C**reation and **A**nalysis **T**ool), have been created to provide the technique and tool support for the safety analysis of safety-critical software product lines. The PL-SFTA allows for the identification of new safety requirements and the analysis of safety-critical requirements and requirement interactions. An AOSE-adapted Software Failure Modes, Effects and Criticality Analysis (SFMECA) technique has been created to support the derivation of a safety analysis asset using the specifications of Gaia-PL allowing for the identification of possible hazard scenarios and the failure points of specific agent roles. Using the assets generated via PL-SFTA and SFMECA, Bi-Directional Safety Analysis (BDSA) is shown to aid in the completeness of PL-SFTA and SFMECA, help verify the safety properties and strengthen the safety case when safety compliance to safety standards of the multi-agent system is necessary.

Results from an application to a large, safety-critical, multi-agent system product-line show that Gaia-PL provides strong reuse capabilities. Evaluation of the Gaia-PL methodology used in conjunction with the PL-SFTA, SFMECA and BDSA safety analysis techniques shows that safety analysis of an agent-based software system is feasible, reusable and efficient.

CHAPTER 1. INTRODUCTION

Safety-critical, agent-based systems are being developed without mechanisms and analysis techniques to discover, analyze and verify software requirements and prevent potential hazards. Agent-oriented, software-based approaches have provided powerful and natural high-level abstractions in which software developers can understand, model and develop complex, distributed systems. Yet, the realization of agent-oriented software development partially depends upon whether agent-based software systems can achieve reductions in development time and cost similar to other reuse-conscious software development methods. Further, agent-oriented software engineering (AOSE) currently does not adequately address: (1) requirements (specification) reuse in a way that is amenable to the reduction of the development cost by utilizing reusable assets, and (2) analysis techniques to evaluate safety.

This dissertation addresses these problems by developing an AOSE methodology, Gaia-PL, that can reduce the cost of developing an agent-based system by producing and utilizing reusable assets during the requirements (specification) phase of design and development. Further, this dissertation details several product-line oriented, safety analysis techniques that can evaluate the safety of an agent-based, product-line system in such a way that: (1) discovers, verifies and analyzes the agent-based systems' requirements, and (2) produces safety analysis assets that are reusable for other agent-based systems created within the same product line.

The work presented here is part of a larger effort that investigates how safety analysis can become a reusable asset of a product line by developing a framework and a suite of techniques and tools for the safety analysis of product lines. The long-term goal is to provide verification results for a new system in the product line in a timely and cost-efficient manner.

This chapter begins with the motivation for this work and an overview of the contributions of this dissertation. First, software product-line engineering is discussed as an incentive for its extension to AOSE to develop multi-agent system product lines (MAS-PL). Next, software safety analysis for safety-critical, software product lines is discussed as a driving factor for the development of techniques and tools tailored to the development of reusable safety analysis assets for product lines. Then, motivation for the inclusion of such product-line safety analysis techniques into the development of MAS-PL is provided. The introduction concludes by stating my thesis and providing an outline for the remainder of the dissertation.

1.1 Product-Line Engineering for Agent-Based Systems

Reuse is highly desirable in software engineering as a way to reduce the cost of the design and development of software. Approaches to achieve reuse have been pursued implicitly and explicitly in the design and development of software systems for many years [12], [76]. For example, software design patterns have been proposed as a design template that acts as a repeatable solution for commonly occurring problems in software design [33]. Object-Oriented Programming has been widely used as an approach to reuse logical units of software code in several different applications [33].

Implicitly, software programmers commonly copy existing code into a new application when the functionality is similar [61]. The product family concept was first introduced by Parnas in [61]. Parnas's claim is that it is advantageous to study a set of programs when the programs share many common features. When developing a set of programs that share common features, Parnas suggested that it is best to initially identify those features that are common to all the programs and then modify and accommodate the design. This produces tailored programs as the leaves of a tree structure where the nodes within the tree represent the design decisions made to arrive at a leaf node.

Software reuse technologies have been a driving force in significantly reducing both the time and cost of software requirements specification, development, maintenance and evolution [11], [12], [67], [74], [88]. Industry's continuous demand for shorter software development cycles and lower software costs encourages software development methodologies to exploit software reuse principles whenever possible.

Software product-line engineering is one such reuse technology that supports the systematic development of a set of similar software systems by understanding, controlling and managing their common, core characteristics and their differing variation points [12], [67]. Software product-line engineering models provide software engineers with a reuse-conscious development platform that can contribute to significantly reducing both the time and cost of software requirements specification, development, maintenance and evolution [12]. In a product line, the common, managed set of features shared by all members is the commonalities. The members of a product line may differ from each other via a set of allowed features not necessarily found in other members of the product line (i.e., the variabilities). The benefits of the product-line concept come from the reuse of the common requirements of the product line in the development of a new product-line member [76]. Software product-line engineering is further discussed in Chapter 2 as related and background work to the provided in this dissertation.

Agent-oriented, software-based approaches have provided powerful and natural high-level abstractions in which software developers can understand, model and develop complex, distributed systems [90], [92], [94]. Yet, the realization of agent-oriented software development partially depends upon whether agent-based software systems can achieve reductions in development time and cost similar to other reuse-conscious software development methods such as object-oriented design, service-oriented architectures and component based systems [7].

In recent years, several Agent-Oriented Software Engineering (AOSE) methodologies have been proposed for various agent-based application domains. The Gaia methodology [92], [94], in particular, offers a comprehensive analysis and design framework based on organizational abstractions by supplying schemas, models and diagrams to capture the requirements of an agent-based software system.

The Gaia methodology centers on defining an agent based upon the role(s) that it can assume during its lifetime [92], [94]. Each role's requirements specification is defined by its protocols (i.e., defines how agents interact), activities (i.e., the computations associated with the role that can be executed without interacting with other agents), permissions (i.e., the information resources that the role can read, change and generate) and responsibilities (i.e., the liveness and safety properties the role must ensure).

However, Gaia has three limitations. First, although Gaia provides a mechanism to allow the role of an agent to change dynamically, it is unclear how to document agent requirements specifications during the analysis and design phases when an agent must be updated to include new functionality. Second, the design of an agent in Gaia is not hierarchical [42]. That is, the roles of an agent are coarsely defined, allowing little flexibility (i.e., little opportunity for reuse) for similar, yet slightly different behavior in the same role in different agents. Third, the Gaia methodology fails to provide a mechanism by which the requirements specification templates developed during the analysis phase can be reused to be incorporated into the current system or to build a new, similar but slightly different system.

This dissertation offers our AOSE methodology, Gaia-PL (Gaia – Product Line) for open, agent-based distributed software systems to capture requirements specifications that can be easily reused during the initial requirements phase as well as later if the software needs to be updated. Our methodology uses a product-line perspective to

promote reuse in agent-based, software systems early in the development lifecycle so that software assets can be reused in the development lifecycle and during system evolution.

The main contribution of this work is a requirements specification pattern to capture the dynamically changing design configurations of agents and reuse the requirement specifications for future similar systems. The ability of the requirements specifications to accommodate the dynamically changing design configurations of an agent is important because an agent may need to adapt and reconfigure itself based on external conditions (e.g., environment conditions, state of the MAS, changing goals, etc.). This is achieved by adopting a product-line approach into AOSE. *Requirements specifications reuse* is the ability to easily use previously defined requirements specifications from an earlier system and apply them to a new, slightly different system. This can significantly reduce the development time and cost of building an agent-based system.

Specifically, the following are contributions of the Gaia-PL methodology work that will be detailed in this dissertation:

- The inclusion of software product-line engineering principles into the development of MAS to build MAS product lines (MAS-PL) [19]
- The creation of an AOSE methodology, Gaia-PL, that supports the design and development of MAS-PL using aspects of Gaia, an established AOSE methodology, and FAST, an established software product-line engineering methodology [19], [21]
- The illustration of how our Gaia-PL methodology is amenable to the development of reusable software engineering assets during the design and development of MAS-PL and how the reusable assets can be used to develop systems of the MAS-PL [19], [21]

- An evaluation of our Gaia-PL methodology's ability to reduce the development cost of MAS via a case study and comparison to the Gaia methodology

This dissertation details the development of an agent-based software product line using our AOSE methodology, Gaia-PL to illustrate its ability to reuse produced software engineering assets and reduce the effort needed for the development of such a system. We demonstrate our approach on an agent-based, software product line – NASA's Prospecting Asteroid Mission (PAM). Although this dissertation illustrates our Gaia-PL AOSE methodology using PAM as a case study, our prior work has shown this methodology applied to another NASA-proposed mission, the TechSAT21 mission [8], [71], [85], in [18], [19], [21] and [22].

Chapter 4 provides the application of Gaia-PL on the PAM case study. The next section further motivates the need for safety analysis in AOSE and discusses our additional work in this area.

1.2 Safety Analysis for Safety-Critical Software Product Lines

Reusability has transformed entire industries and caused software engineers to adapt their methods to further this goal. The software product-line engineering approach supports reuse by developing a suite of products sharing core commonalities [12]. However, the development of safety-critical, software product lines in industry has emerged ahead of the development of product-line, safety analysis techniques and tools. This has created a lack of techniques and tools available to software engineers to ensure the safe reuse of software engineering artifacts throughout a product line [51]. It is only after a full suite of safety analysis tools and techniques are available to software engineers to ensure the safety in safety-critical product lines that safety-critical software

product lines will gain organizational and industrial acceptance and assume more responsibility in everyday safety-critical applications.

Performing safety analysis on software product lines previously entailed considering each product line member in isolation and applying traditional safety analysis techniques to them [44]. Yet, this fails to leverage the fact that product-line members share a common core.

This dissertation offers additional assurance to software engineers by providing a safety analysis technique applicable to product lines. Specifically, an adaptation of the Software Fault Tree Analysis (SFTA) technique applied to product lines in order to derive reusable analysis assets for future systems within the existing product line is detailed [17]. The product-line SFTA (PL-SFTA) maintains the safety analysis qualities of traditional SFTA while accommodating the reusable asset objective of the product-line concept. Traditional SFTA targets the safety analysis of potentially harmful states for one specific product. A PL-SFTA, however, contributes to the safety analysis for the entire product line including variabilities among the products. The PL-SFTA can then be reused as part of the safety analysis for the introduction of new product line members. The development of the SFTA for the new product is achieved through a pruning method. The goal is to support the reduction of the safety analysis needed on a new product within the product line and, ultimately, a less expensive and shorter product development process.

This dissertation provides a detailed process by which a software engineer can construct a PL-SFTA for the initial product line, derive the new system's SFTA from the PL-SFTA and modify the PL-SFTA to accommodate changes in requirements due to system evolution of a product line. In addition, this research has provided mechanisms that:

- Aid in discovering additional system safety requirements [17]
- Help in identifying additional product-line dependencies [18]

- Allow for analyses to assess failure points and safety critical requirements [23]
- Complement Software Failure Modes, Effects and Criticality Analysis, Bi-Directional Safety Analysis and other safety analysis techniques to strengthen a safety case when system certification is required [22]

To support this technique, a software safety analysis tool, called PLFaultCAT (**P**roduct-**L**ine **F**ault Tree **C**reation and **A**nalysis **T**ool) has been developed as a part of this work. This tool builds on a previously developed technique that adopted Software Fault Tree Analysis (SFTA) to product line safety analysis [17]. PLFaultCAT is an interactive, partially-automated software support application to aid software engineers with the visualization and pruning process of a PL-SFTA. Specifically, the tool exploits the reusability inherent in product-line engineering by deriving reusable safety analysis assets (i.e., the product-line members' fault trees) for future systems within the existing product line.

The contribution of this work is to further investigate how and to what extent the PL-SFTA technique, supported by the PLFaultCAT tool, can be used by software engineers as a reusable safety analysis. This approach employs Weiss and Lai's Family-Oriented Abstraction, Specification, and Translation (FAST) model [88]. This model employs a two-phase software engineering approach: the domain engineering phase and the application engineering phase. The domain engineering phase defines the product line and constructs the PL-SFTA with the aid of the PLFaultCAT tool; the application engineering phase develops and performs the safety analysis on new product-line members also using PLFaultCAT.

We first provide a framework for the construction, aided by PLFaultCAT, of a PL-SFTA during the domain engineering phase and then supply the means for reusing the PL-SFTA for new members as it is implemented in the PLFaultCAT tool. Within the

application engineering phase we utilize PLFaultCAT to facilitate the derivation of new product-line members' fault tree(s).

In addition, the main contributions of the PLFaultCAT tool to support our PL-SFTA safety analysis technique described in this dissertation include:

- Automatically derive all of the product line member SFTAs from PL-SFTAs [24]
- Link product-line requirements to PL-SFTA nodes to aid in traceability [23]
- Search the set of PL-SFTAs to identify single-point failures [18]
- Identify safety-critical requirements by analyzing the set of PL-SFTAs [18], [48]
- Provide a minimum-cut set analysis of a PL-SFTA to identify hazard paths [24]

This dissertation details each of these contributions for an agent-based, software product line – NASA’s Prospecting Asteroid Mission (PAM). This dissertation illustrates our PL-SFTA technique using PLFaultCAT for an agent-based system by extending our prior work which applied this technique and tool to Weiss and Lai’s [88] Floating Weather Station in [17] and [24], to a pacemaker product line in [45], [47], [48] and to another NASA-proposed mission, the TechSAT21 mission [8], [71], [85], in [18], [19], [21] and [22].

Chapter 5 reports on this work, as well as additional safety analysis techniques we have adopted for the use in our Agent-Oriented Software Engineering (AOSE) methodology, Gaia-PL (Gaia – Product Line). The next section further motivates the need for safety analysis in AOSE and discusses our additional work in this area.

1.3 Safety Analysis for Multi-Agent System Product Lines

Safety-critical systems composed of highly similar, semi-autonomous agents are being developed in several application domains. An example of such a multi-agent system (MAS) is a swarm of satellites. In swarms of satellites, each satellite is

commonly treated as a distinct autonomous agent that must cooperate to achieve higher-level goals of the swarm [71].

The emergence of distributed systems (e.g., formation-flying, satellite swarms) as a viable and reliable architecture for mission-critical domains coupled with the advantages of adopting an agent-oriented perspective for software development has led to a number of proposed systems utilizing these two concepts. A MAS is an application “designed and developed in terms of autonomous software entities that can flexibly achieve their objectives by interacting with one another in terms of high-level protocols and languages” [94].

Actual proposed systems including the Terrestrial Planet Finder-I (TPF-I) spacecraft [81] and the TechSat-21 [8], Sun-Solar System Connection, Search for Earthlike Planets and Universe Exploration all rely on constellation missions to achieve their scientific goals [56]. In addition to these examples, there is NASA’s Prospecting Asteroid Mission [71], [77], [83], [84], the case study used throughout this dissertation and detailed in Chapter 3. Agent-oriented software engineering (AOSE) appears to be an appropriate software development methodology for such systems [71].

A safety-critical system can directly or indirectly compromise safety by placing a system into a hazardous state causing the potential loss or damage of life, property, information, mission or environment [44]. Like other safe-critical software systems (e.g., cardiac pacemakers, aircraft flight-control systems, military weapons systems, nuclear power monitoring systems, etc.) some MAS require extensive safety analysis and, potentially, safety certification. Although scientific satellite swarms, such as the PAM case study used in this dissertation, may not directly cause the loss of human life as a result of an accident, a system-wide failure/accident may result in the loss of an entire mission, the spacecraft and the millions of dollars of investment.

A challenge to safety analysis of multi-agent distributed systems, such as constellations of satellites, is the ability of agent-based software systems to dynamically alter their configurations (for example, from active to passive). A configuration of an agent in this work is the set of behaviors implemented in an agent's roles. In addition, we would like to reuse safety analysis results while ensuring the maintenance of safety. That is, a tradeoff of higher reuse potential for less safety in the final product is not acceptable.

Certification is a process whereby a certification authority determines if an applicant provides sufficient evidence concerning the means of production of a candidate product and the characteristics of the candidate product so that the requirements of the certifying authority are fulfilled [31], [40], [69], [72]. Software safety analysis techniques have previously been shown to contribute to the certification of software-intensive systems in [2]. However, little work has been specifically aimed at software product lines or MAS. In addition to illustrating our product-line Software Fault Tree Analysis (PL-SFTA) for a MAS product line (MAS-PL), described in the previous section, this dissertation adopts and tailors additional safety analysis techniques of our AOSE methodology, Gaia-PL (Gaia – Product Line), to support the creation of reusable safety analysis assets; discover, verify and analyze safety requirements; and aid in the certification of MAS-PL.

The main contribution of this work is to extend Bi-Directional Safety Analysis (BDSA) to MAS-PL and show how the analysis artifacts thus produced contribute to the software's safety case for certification purposes. The product-line approach lets us reuse portions of the safety analysis assets for multiple, similar agents, significantly reducing the burden of certification.

First, we further the inclusion of safety analysis techniques into AOSE by providing a structured process to perform a Software Failure Modes, Effects, Criticality Analysis (SFMECA) for safety-critical MAS-PL in our Gaia-PL methodology. The

SFMECA is reusable for other agents in the system since our approach incorporates the product-line vision of a MAS from [21].

Second, we use the safety analysis assets from SFMECA and from our product-line Software Fault Tree Analysis (PL-SFTA), described in the previous section, to perform a BDSA on the MAS-PL to contribute to system certification by verifying software design compliance with robustness and safety standards. The application of BDSA to a MAS-PL assists in the certification of agent-based software systems by:

- Providing assurances that certain classes of failure modes that might occur in individual agents will not produce unacceptable effects in the composite system, strengthening the safety case by demonstrating the compliance of failure-monitoring and failure mitigation software tasked with the system safety requirements to safety standards
- Enabling reuse of certification arguments while ensuring that the reuse of the safety analysis artifacts in the certification arguments accurately reflects the differences amongst the agents of the system

This dissertation details this work, along with our PL-SFTA safety analysis technique, as safety analysis techniques for an agent-based, software product line – NASA’s Prospecting Asteroid Mission (PAM), detailed in Chapter 3. This work has been previously been demonstrated on another NASA-proposed mission, the TechSAT21 mission [8], [71], [85], in [18], [19], [21] and [22]. However, the application of our safety analysis techniques for a MAS-PL described in this dissertation is at a much larger scale. Specifically, the PAM MAS-PL case study discussed in this dissertation consists of 97 high-level requirements, including 47 features allowing the development of 160 unique spacecraft in the PAM MAS-PL.

Chapter 5 describes this work using our Agent-Oriented Software Engineering (AOSE) methodology, Gaia-PL (Gaia – Product Line) on the PAM case study.

The next section formally provides this dissertation's statement of thesis and provides the contributions of this dissertation to support the thesis.

1.4 Statement of Thesis

The problems addressed in the work described in this dissertation are twofold. First, Agent-Oriented Software Engineering (AOSE) currently does not adequately address requirements (specification) reuse in a way that is amenable to reducing the development costs (i.e., time and money) by developing and utilizing reusable assets. Second, safety-critical, multi-agent systems (MAS) are being developed without the mechanisms and analysis techniques and tools in AOSE methodologies to discover, verify and analyze software requirements and potential safety hazards.

Based on this problem statement, the theses of the work presented in this dissertation is that *an AOSE methodology can be devised to enhance the reuse in the design and development of a safety-critical MAS by incorporating software product-line engineering principles to develop reusable software engineering assets in a way that allows software engineers to take advantage of the reusable assets to create MAS; and that product-line safety analysis techniques and tools can be developed and adopted to support the development of a safety-critical MAS by discovering, analyzing and verifying the MAS's requirements in a way that produces reusable safety assets that can be used for future systems of the MAS.*

This thesis is supported in this dissertation by:

- Incorporating software product-line engineering principles into the development of MAS to build MAS product lines (MAS-PL)
- Creating an AOSE methodology, Gaia-PL, that supports the design and development of MAS-PL using aspects of Gaia, an established AOSE

methodology, and FAST, an established software product-line engineering methodology

- Illustrating how our Gaia-PL methodology is amenable to the development of reusable software engineering assets during the design and development of MAS-PL and how the reusable assets can be used to develop systems of the MAS-PL
- Evaluating our Gaia-PL methodology's ability to reduce the development cost of MAS via a case study and comparison to the Gaia methodology
- Developing the product-line Software Fault Tree Analysis (PL-SFTA) technique to support the safety analysis of safety-critical software product lines in a way that the resulting PL-SFTA is reusable for the products in a product line
- Designing a software tool, PLFaultCAT, to support the creation of a PL-SFTA and the automatic derivation of a SFTA for the products in a product line
- Evaluating PL-SFTA and PLFaultCAT's ability to reduce development costs through the reuse of the PL-SFTA
- Adapting Software Failure Modes, Effects and Criticality Analysis (SFMECA) into the Gaia-PL AOSE methodology to provide a structured process in which software engineers can derive a SFMECA directly from the assets of our Gaia-PL methodology
- Describing how the PL-SFTA and SFMECA can be used with a Bi-Directional Safety Analysis (BDSA) to discover new/missing safety requirements, verify the safety analyses and contribute to the safety case of a safety-critical MAS

These results, as well as additional contributions to support the thesis statements, are described in the remainder of this dissertation.

1.5 Outline

Chapter 2 reviews related work in software product-line engineering, Agent-Oriented Software Engineering (AOSE) and software safety analysis to provide the necessary context and background information. We additionally discuss the differences of the related work from the work presented here.

Chapter 3 describes the Prospecting Asteroid Mission (PAM) case study that is used throughout this dissertation to illustrate and evaluate our work. This chapter provides the background information needed to understand the domain and context of the case study.

Chapter 4 details our Gaia-PL (Gaia – Product Line) AOSE methodology for designing and developing multi-agent system product lines (MAS-PL). The methodology produces reusable software engineering assets so that building systems of the MAS-PL can be done efficiently, in terms of development cost and time. We evaluate Gaia-PL's ability to reduce the development cost of a MAS through its application to the design and development of a case study, and its comparison to using a different, non-product line, approach.

Chapter 5 discusses our safety analysis techniques and tools for the analysis of safety-critical software product lines. Specifically, this chapter describes our product-line software fault tree analysis technique (PL-SFTA) and its tool, PLFaultCAT. We again provide an evaluation of these safety analysis techniques through an application to our case study to illustrate their value as reusable safety assets, ability to increase safety by identifying new and missing safety requirements and potential for reducing development costs compared to a non-product line safety analysis approach.

Finally, Chapter 6 offers conclusions, a discussion of the research's contributions and ideas for future work.

CHAPTER 2. RELATED WORK

The work described in this dissertation builds upon the overlapping areas of software product-line engineering, agent-oriented software engineering and software safety analysis. This chapter discusses the background information and related work in these areas of software engineering and describes related concepts, techniques, methodologies and tools that are related to the Gaia-PL (Gaia-Product Line) methodology and product-line software safety techniques developed in this work.

2.1 Software Product-Line Engineering

A software *product line* is defined as “a set of software-intensive systems sharing a common managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” [12]. The members of a particular product line differ from each other via a set of allowed variabilities/variation points.

Software *product-line engineering* is a proactive and systematic approach for the design and development of software applications to create an array of similar products [12]. Software product-line engineering creates a family of products and relies on the analysis of the commonalities and differences of the members of the family prior to the design or development of any software engineering artifacts (i.e., during the requirements engineering phase) [87]. The goal of software product-line engineering is to support the systematic development of a set of similar software systems through by understanding, controlling and managing their common, core characteristics and their differing variation points [12], [67].

Software product-line engineering is a widely accepted and active research field in academia. Several academic textbooks solely dedicated to software product-engineering exist including [12], [36], [67] and [88]. In addition, the *Software Product*

Line Conferences [75], the major conference of software product-line engineering, is currently in its 11th cycle including more than a half-dozen associated product-line workshops.

Product-line engineering is also widely accepted and used in industry, and in fact, has been used for many years. For example, automobiles, airplanes, televisions, cellular phones, etc. are product lines that people encounter in daily life. Software product lines have also begun to be adopted by industry. For example, the Software Engineering Institute (SEI) at Carnegie Mellon University has recognized Hewlett Packard [82], Nokia [39], Boeing [27], Philips [89], CelsiusTech Systems [12] and others as members of their Product Line Hall of Fame [68]. These companies, and countless more, have recognized the advantages of software product-line engineering and have adapted its approach as their development paradigm to offer customers a wide variety of products while incurring reduced development effort.

The benefits of the product-line concept come from the reuse of the common requirements of the product line in the development of a new product-line member [76]. Thus, the assets gained from the initial engineering of the product line, such as the underlying architecture, requirements and safety analyses and testing artifacts, can be at least partially applied to any new product-line member. For example, CelsiusTech Systems claims to have reused up to 90% of their assets in the development of systems in their shipboard command and control systems product line [12]. In this sense, product line engineering allows for the amortization of costs in startup development and analysis of the initial product line members over the development of the entire product line. In fact, studies suggest that the product-line engineering concept can reduce the development and production time as well as the overall cost and increase the product quality by a factor of 10 times or more [74].

The following subsections review the software product-line engineering terms, techniques and tools relevant to this work.

2.1.1 Commonality and Variability Analysis

The analysis, design and documentation of commonality and variability requirements play a crucial role in all phases of software product-line engineering [12]. Early in the development of a software product line, a product line's requirements are often identified and specified through a Commonality and Variability Analysis (CVA). The CVA, as detailed by Ardis and Weiss in [1] and Weiss and Lai in [88], provides a comprehensive definition of the product line that provides a dictionary of terms, a list of the commonalities, a list of the variabilities and a list of parameters of variation. Although not a part of the CVA as detailed in [1] and [88], a list of dependencies/constraints on the variabilities of the product line may also be included. The CVA technique aids in providing a software engineering artifact that details the relevant domain definitions, the core set of product traits and the scope of the product line.

2.1.1.1 Commonalities

Pohl, Böckle and van der Linden define a product line *commonality* as a requirement that is identical in each member of a family [67]. Similarly, Weiss and Lai define a commonality as an “assumption that is true for all members of a family” [88]. Commonalities describe requirements of the entire product line and contribute to the development of the core assets of the product line that are common to all members of the product line. An example of a product-line commonality is “All widgets of the WidgetFamily product line shall have four wheels”. This means that, any product that is built from the WidgetFamily product line must, without exceptions, have four wheels.

Any product that does not abide by this requirement is, by definition, not a member of the product line since it does not have this product-line commonality.

2.1.1.2 Variabilities

Weiss and Lai define a product-line *variability* as “an assumption about how members of a family may differ from each other” [88]. Variabilities capture optional or alternative features not contained in every member of the product line and should capture the anticipated variations of the product-line member over the “foreseeable lifetime of the product line” [12]. An example product-line variability is “The color of the products in the WidgetFamily product line may vary”.

Variabilities also frequently have associated "parameters of variation" that detail the degree to which the variability can occur [88]. The parameters of variation describe the acceptable range of variation. Weiss and Lai describe the *parameters of variation* as a “quantification of a variability, including the decision represented by the variability, the range of values allowed in making the decision, the time at which the value for the decision must be fixed, and a default value for the decision” [88].

A variability's parameters of variation within a product line often fall into one of three categories: Boolean parameters of variation, enumerated parameters of variation, or range parameters of variation. These categories of parameters of variation get increasingly more difficult to analyze for safety as the complexity in the number of choices increases. Boolean parameters of variation are those variabilities that can either be present within a product-line member or not. An enumerated parameter of variation is any variability in which the product-line member must choose from a relatively small list of values for a particular variability.

A simple example of an enumerated parameter of variation is "Widget X can either be blue, green, red, or yellow". A range parameter of variation are those

variabilities in which the product-line member must have a precise number associated with the variability, where the number lies within the range of acceptable parameters of variation specified in a Commonality and Variability Analysis (CVA). For example, "Widget X may have between 1 and 100 user functions" is a simple range parameter of variation.

2.1.1.3 Dependencies

A product-line *dependency* (i.e., constraint) restricts and/or dictates some combinations of variability subsets from being viable products in the form of "mutual exclusion" or "requires" variability dependencies [28], [43]. A dependency requirement can thus take the form "Any product-line member that has Variability A can not also have Variability B" or in the form "Any product-line member that has Variability C must also have Variability D". The first example indicates that any member of this product line is restricted from displaying both behaviors A and B. Alternatively, a dependency may also be in the form "Any product-line member that has Variability A with a value of 'a' can not also have Variability B with a value of 'b'". Dependency requirements can derive from actual physical limits, undesired or infeasible combinations of behaviors, user restrictions, or business decisions.

Dependency requirements are especially important for the hazard analysis of a safety-critical product line and should be explicitly documented. By reducing the subset of potential viable products stemming from the product-line definition, we reduce the scope of the needed hazard analysis.

2.1.2 Feature Modeling

An alternative or supplemental approach to defining a software product line is in terms of its mandatory (i.e., required), optional (i.e., not required) and alternative (i.e., one or more from a list of alternatives is required) features [36], [67]. Svahnberg, Gulp

and Bosch define a feature as a “logical unit of behavior that is specified by a set of functional and quality requirements” [80]. A feature model hierarchically defines the mandatory, optional and alternative features of a product line by breaking down a single, high-level feature into its subfeatures. A product of a product line is thus a set of the mandatory features, a selection amongst the alternative features and the desired optional features. A child feature can only be present in a product of the product line if its parent feature is also present.

2.1.3 Product-Line Engineering Phases

Weiss and Lai’s Family-Oriented Abstraction, Specification and Translation (FAST) approach is an approach for developing product families that was designed and used at Lucent Technologies [88]. The FAST approach is based on investing resources proactively in the early design of a set of systems to identify their common and variable parts [88], as advocated by Parnas in [61]. The FAST approach advocates such a strategy because they claim that the high investments of resources in the early design stages are amortized over the set of product-line members that are produced. Similarly, the time-to-market and variety in the production of new products within the product line will provide the company with a competitive advantage [88].

The FAST approach for building software product lines utilizes the Commonality and Variability Analysis (CVA) and partitions the design and development of a product line into two unique phases: *domain engineering* and *application engineering*.

2.1.3.1 Domain Engineering

Pohl, Böckle and van der Linden define *domain engineering* as “the process in software product-line engineering in which the commonality and the variability of the product line are defined and realized” [67]. The goal of the domain engineering phase of the FAST approach is to define the product-line requirements, design, architecture and

other software engineering assets that pertain to the entire product line, rather than a single product-line member [88]. This process relies primarily on the knowledge and skill of domain experts to produce such assets [12], [67]. The purpose is to make it possible to produce members of a product line, during the application engineering phase, using the assets generated during this phase. This is the investment phase that allows practitioners to, during the application engineering phase, quickly realize a wide variety of products within the product line for a competitive advantage.

2.1.3.2 Application Engineering

Pohl, Böckle and van der Linden define *application engineering* as “the process of software product-line engineering in which the applications of the product line are built by reusing domain artifacts and exploring the product-line variability” [67]. The goal of the application engineering phase of the FAST approach is to build an individual product-line member(s) from the product-line requirements specified during the domain engineering phase [88]. Building a new product in the product line during this phase entails selecting values for all the parameters of variation consistent with the dependencies as detailed in the Commonality and Variability Analysis (CVA).

2.1.4 DECIMAL

Given a product-line’s commonalities, variabilities and dependencies as detailed in a Commonality and Variability Analysis (CVA), a valid product-line member’s requirements consist of the commonalities and a selection of variabilities (and their values) that conform to the dependencies for the product line. To aid in automatically checking the conformance of a product-line member to a product line’s variabilities and dependencies, Padmanabhan and Lutz developed DECIMAL (**D**ecision **M**odeling **A**pplication), a requirements validation tool to certify that a set of requirements for a proposed product line member does not breach the dependencies of the product line [58],

[59]. In addition, DECIMAL provides requirements engineers with the ability to document the commonalities, variabilities and dependencies of a product line, define a variability in terms of its parameters of variation and define a product-line member through the choice of its variabilities (and their values) [23], [58], [59].

2.1.5 Summary

Software product-line engineering is a rich, established software design and development field that has been shown to be advantageous. This approach relies on the development of reusable, core assets that can be used in the design of a set of similar, yet differing, software systems. The use of such reusable, core assets has been shown to provide significant cost savings (i.e., development time and cost) to the development of a set of software systems.

The work described in this dissertation integrates the software product-line engineering approach described in this section to the development of agent-based software systems, described in the following section. In addition, this dissertation develops techniques and tools for the safety analysis of software product lines built using the concepts and approach described in this section.

2.2 Agent-Oriented Software Engineering

The second area of related research that this dissertation draws on is the increasingly important design and development of distributed, agent-based software systems. In addition to the increase in complexity [40], the demands and expectations placed on modern software systems have significantly changed causing a new set of challenges to be addressed by software engineering [94]. Zambonelli, Jennings and Wooldridge argue the following new characteristics of today's software systems:

- By default are concurrent, distributed and expected to interact with components and services that are dynamically discovered at runtime

- “Always-on” entities that can’t be stopped, maintained or restored in the traditional ways
- Exist in an open, dynamic environment where new component join, existing components leave and the operating environmental conditions change in a, possibly, unpredictable manner

One approach to address and accommodate these new challenges in software engineering is Agent-Oriented Software Engineering (AOSE) [40], [90], [92], [94]. The AOSE methodology presented in this dissertation, Gaia-PL (Gaia - Product Line) differs from Gaia in that we integrate software product-line engineering concepts into the Gaia methodology allowing for software engineering to capture the reuse potential of a MAS’s software engineering assets so that future systems can be built quickly and cheaply.

The following subsections review the AOSE terms, techniques and tools relevant to this work.

2.2.1 Agents and Multi-Agent Systems

Agent-Oriented Software Engineering (AOSE) designs and develops the agents of a multi-agent system (MAS) to solve a problem. Wooldridge defines an *agent* as “an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives” [90]. Jennings and Wooldridge [40] and Zambonelli, Jennings and Wooldridge [94] classify several characteristics that comprise an agent’s behavior:

- Problem solving entities that are clearly identifiable, have well-defined boundaries and interfaces
- Situated in a particular environment where they receive inputs related to the states of the environment via sensors and may act upon the environment via their effectors

- Have specific objectives (i.e., roles) to achieve that may be explicitly or implicitly represented within the agent
- Autonomous in that they have control of their internal state and their own behavior
- Capable of exhibiting flexible, context-dependent, problem-solving characteristics
- Able to respond to changes that occur in their environment in a timely manner so that they can satisfy their objectives (i.e., reactive)
- Able to opportunistically adopt new objectives whenever appropriate and take the initiative to satisfy their goals (i.e., proactive)

For most problems, a single-agent solution is insufficient and thus requires a multiple-agent solution [40]. Zambonelli, Jennings and Wooldridge define a *multi-agent system* (MAS) as an application that is “designed and developed in terms of autonomous software entities that can flexibly achieve their objectives by interacting with one another in terms of high-level protocols and terms” [94]. Thus, AOSE’s objective is to provide software engineers with the design methodologies to develop the agents of a MAS to address the solution of a particular problem.

2.2.2 Agent-Oriented Software Engineering Methodologies

Agent-oriented software engineering (AOSE) [91] methodologies surfaced in the late-90’s to provide tools and techniques for abstracting, modeling, analyzing and designing agent-based software systems early in the development lifecycle [79]. Different methodologies, such as Gaia [6], [92], [94], Tropos [3], [34] and MaSE [25] for example, use different abstractions and models for agent-oriented software development. Recent work has produced AOSE methodologies that focus on the reusability of software engineering assets produced so that future systems can be developed faster and cheaper.

This section briefly details these methodologies in the context of the work presented in this dissertation.

2.2.2.1 The Tropos and MaSE Methodologies

The Tropos Agent-Oriented Software Engineering (AOSE) methodology covers all phases of software development (from early requirements engineering to actual implementation) and is based on the notions that agents have goals and plans [3], [34]. Based on Yu's *i** goal-modeling approach [93], Tropos focuses on developing and understanding the goals and subgoals of a system and the agents of a system through the creation of goal model diagrams [3], [34].

The Multiagent Systems Engineering (MaSE) AOSE methodology utilizes graphically based models derived from standard UML models to analyze the agents of a software system [25]. Unlike Tropos, MaSE views the development of a multi-agent system (MAS) as a further abstraction of the object-oriented (OO) paradigm and, thus, it builds upon known OO techniques and applies them to the design and development of MAS.

The work described here differs from both Tropos and MaSE previous work in that our methodology focuses on developing reusable assets rather than understanding and developing the goals and subgoals of a MAS as in Tropos or adapting UML models for MAS as in MaSE.

In addition, we adapt the Gaia AOSE methodology (discussed in the next section) for the development of MAS rather than using Tropos, MaSE or any other AOSE methodology.

2.2.2.2 The Gaia Methodology

The Gaia Agent-Oriented Software Engineering (AOSE) methodology was the first methodology proposed in literature to guide the process of designing and developing

a multi-agent system (MAS) from the analysis phase to the design phase [6], [92], [94]. The Gaia methodology adopts a computational organizational metaphor where each agent within a MAS may play a variety of roles and where the agents cooperate with each other to accomplish a common organizational goal [6], [94].

Briefly, the analysis phase of the design and development of a MAS in Gaia methodology, shown in Figure 1 [94], concentrates on specifying the requirements and specifications of the roles that an agent may participate during its lifetime in a set of Role Schemas. An Agent Model defines an agent by associating the roles, detailed in the Role Schemas, that that agent may partake in.

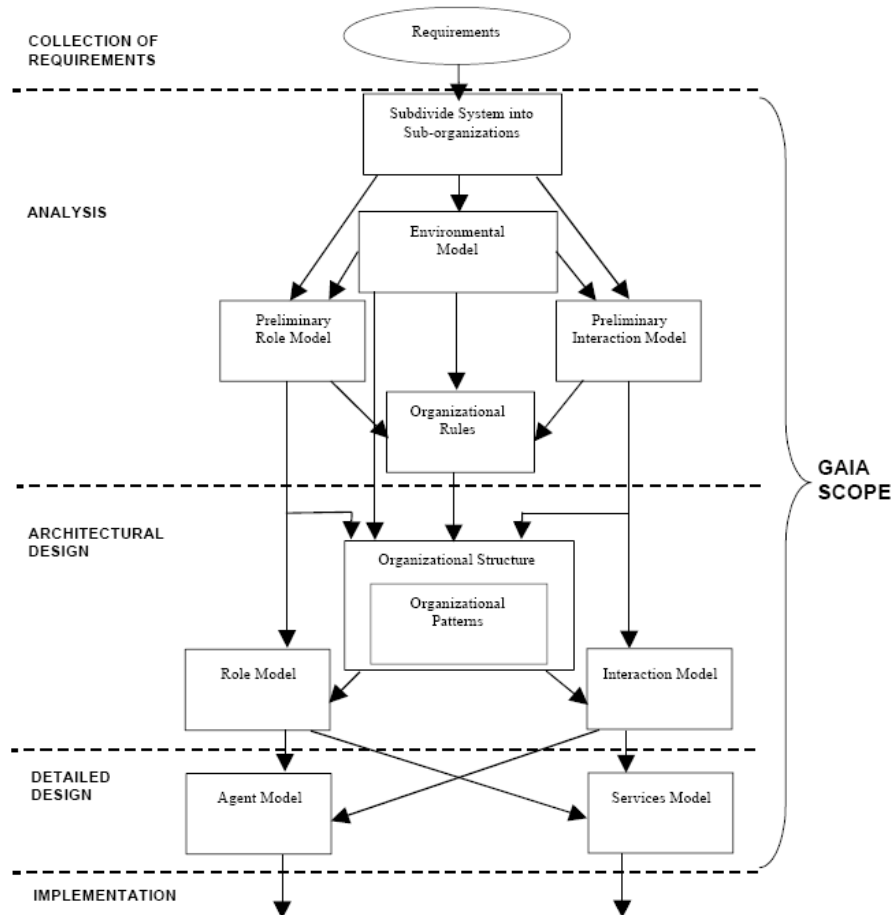


Figure 1 The Gaia Models and their Relationships (from [94])

The Gaia methodology was selected in this work for several reasons. First, it was the most mature AOSE methodology (i.e., it spans from the analysis phase to the design phase of agent-based development). Second, it is an established, well-published and widely accepted methodology in the AOSE community. Finally, the Gaia methodology's development process and models best fit with the phases of the software product-line engineering, described in the previous section.

The AOSE methodology described here, Gaia-PL (Gaia - Product Line) differs from Gaia in that we integrate software product-line engineering concepts into the Gaia methodology allowing for software engineering to capture the reuse potential of a MAS's software engineering assets so that future systems can be built quickly and cheaply. Further, Gaia-PL focuses on capturing the reusability of the software engineering assets developed during the design and development of a MAS using a software product-line engineering approach so that future systems can be built quickly and easily.

2.2.2.3 Reuse-Oriented Methodologies

From its onset, one of the goals of Agent-Oriented Software Engineering (AOSE) has been to provide methodologies for reusing and maintaining agent-based software systems [85]. In spite of this goal, AOSE methodologies have failed to adequately capture the reuse potential, since many of the developed methodologies center on the development of specific software applications [34]. A few attempts, including [34] and [38], have been proposed for reuse in an agent-oriented development environment. However, in each case, reuse is positioned in the later stages of design and development. In [34], the Multi-Agent Application Engineering (MaAE) work exploits reuse during the design phase of a multi-agent software system. Likewise, [38] utilizes reuse principles from component-based development to reuse components from a previously developed agent-based component repository.

The work described here differs from previous work in that we present an approach, based on software product-line engineering, to capture the reuse potential of distributed, agent-based software systems in the requirements analysis and specification stage.

2.2.2.4 The MaCMAS Methodology

More recently, Peña, Hinchey, Ruiz-Cortés and Trinidad developed the Methodology for analyzing Complex Multiagent Systems (MaCMAS) using a software product-line engineering approach to build multi-agent system product lines(MAS-PL) [62], [64], [65], [66]. Their AOSE methodology uses UML to model a MAS-PL and focuses on handling the complexity of MAS-PL and building its core architecture [64].

The MaCMAS methodology, like the Gaia-PL (Gaia-Product Line) methodology described in this work, utilizes a feature model to document the commonalities and variabilities of the MAS [64]. In addition, the MaCMAS methodology uses an automatic algorithm to analyze the features (i.e., commonality and variability requirements) of a MAS-PL to partition the requirements as either commonalities or variabilities based on the probability that a feature will appear in a product [64]. This information is then used to determine which features should be included, using their approach, in the MAS-PL's core architecture [64]. For example, a feature that is projected to be present in 60% of the products of a particular MAS-PL will be included in the MAS-PL's core architecture.

The MaCMAS methodology incorporates and extends our idea of incorporating software product-line engineering techniques into AOSE originally reported in [19], [21]. The AOSE methodology, Gaia-PL, presented here extends an established, well-known AOSE methodology, Gaia [6], [92], [94], by introducing software product-line engineering concepts from an established, well-known software product-line engineering approach, FAST [88]. Further, the Gaia-PL methodology presented in this dissertation

and previously presented in [19] and [21] differs from that of MaCMAS in that we focus on the reusability of the MAS-PL's requirements, requirements specifications and safety analysis assets rather than the MAS-PL's architecture.

2.2.4 Summary

Agent-Oriented Software Engineering (AOSE) is a growing field in software engineering for the design and development of multi-agent systems (MAS). AOSE methodologies provide software engineers with the techniques to abstract, analyze and design of MAS.

The work described in this dissertation integrates software product-line engineering approach into an AOSE methodology to develop MAS product lines (MAS-PL). In addition, this dissertation develops and integrates techniques and tools for the safety analysis of MAS-PL.

2.3 Software Safety Analysis

A safety-critical system can directly or indirectly compromise safety by placing a system into a hazardous state causing the potential loss or damage of life, property, information, mission, or environment [44]. Safety-critical software systems are being assimilated into our everyday lives in a vast range of domains and markets [51]. Safety-critical software runs applications such as pacemakers, aircraft flight-control systems, military weapons systems and nuclear power monitoring systems. Software safety analysis aims at providing safety and software engineers with the techniques and tools to ensure the safety of such software applications.

Just as autonomous software products have caused accidents, product-line software applications have also contributed to catastrophic losses. For example, the Therac-25 medical system and the Ariane 5 losses were accidents caused, in part, by product-line engineering mistakes [44], [76]. The work described here is particularly

aimed at providing safety analysis techniques for safety-critical product lines to prevent such accidents.

The following subsections review software safety and the software safety analysis techniques relevant to this work.

2.3.1 Software Safety

The aim of software safety is to prevent accidents through the analysis of possible hazards in the software system. Leveson defines *safety* in a software system as “freedom from accident or losses” [44]. An accident is an “undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss” [44]. A hazard is a “state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident (loss event)” [44].

2.3.2 Software Safety Analysis Techniques

Software safety analysis techniques center on the investigation of how software can jeopardize or contribute to the safety of the system [44]. The following subsections describe three of the most common safety analysis techniques used by software engineers on safety-critical software: Software Fault Tree Analysis (SFTA), Software Failure Modes, Effects and Criticality Analysis (SFMECA) and Bi-Directional Safety Analysis (BDSA). These safety analysis techniques are used in the work described in this dissertation and are described next.

2.3.2.1 Software Fault Tree Analysis

Software Fault Tree Analysis (SFTA) is a traditional safety analysis technique that has proven to be an essential tool for software engineers during the design phase of a safety-critical software product [37], [44], [53], [60]. SFTA is a tree-based top-down

(deductive), backward search method utilizing Boolean logic to depict the causal event contributing to an undesirable event (the root node). The analysis begins at the root node with the engineer specifying a root node event. For safety-critical systems, the root node of the tree will often represent a system-wide, catastrophic event taken from a preexisting hazards list [44]. The hazard represented by the root node is hypothesized to have occurred, and the engineer proceeds to determine the set of necessary preconditions causing the root node. The set of possible causes are joined to the parent node by standard logical relations represented via logic gates to describe their contributing relation. This process continues through each level of the constructed subtree until basic events are reached or until the level of subsystem detail is achieved [44].

However, traditional SFTA only considers the behavior of a single system rather than the behaviors of the multiple systems of a product line, as of concern in this work. Coppit and Sullivan in [13] and Pai and Dugan in [60] examine dynamic SFTA to represent multiple possible outcomes of a component failure, for example, depending on whether a cold-spare, warm-spare or hot-spare component is available. However, these approaches still only describe single-system behavior rather than the product-line behavior of concern here.

Lu and Lutz presented the Fault Contribution Tree Analysis (FTCA) approach in [49]. Their approach, closely related to SFTA, analyzes the safety and robustness of safety-critical product lines in a reusable tree-like structure. Using the FTCA approach, software engineers can prune the FTCA for specific product-line members. The Product-Line Software Fault Tree Analysis (PL-SFTA) approach described in this work differs from the FTCA approach in that we adopt the more familiar SFTA for the use with product lines and provide a tool in which developers can create PL-SFTA's and then automatically derive the product-line members' SFTA(s).

2.3.2.2 Software Failure Modes, Effects and Criticality Analysis

Failure Modes, Effects and Analysis (FMEA) is a traditional analysis technique originally developed for reliability engineering to be able to predict equipment reliability with a goal to establish an overall probability that the product will operate without a failure for a certain length of time [44]. Software Failure Modes, Effects and Criticality Analysis (SFMECA) was adopted from FMEA and applied to software-intensive systems. SFMECA is a tabular (inductive), forward-based search technique that starts with the failure of a software component or subsystem and then looks at its effect on the overall system [44]. SFMECA first lists all the components comprising a system and their associated failure modes. The possible causes of failure are listed and the effects on other components or subsystems are evaluated and listed along with the consequence on the system for each component's failure mode(s). Finally, a criticality assessment (e.g., minor, major, critical, catastrophic, etc.) is documented to denote the seriousness of the occurrence of such a failure. Like SFTA, SFMECA is only as good as the domain and system expertise of the analyst. Note that SFMECA and Software Failure Modes and Effects Analysis (SFMEA) are identical except that SFMEA does not evaluate and document the criticality of a failure.

In [53], Lutz and Woodhouse provide a list of generic failure-mode guidewords to aid in the process of constructing a SFMECA for failures in data communication and event processing. These guidewords, when applied to the failure of a component or subsystem, help engineers systematize the process of determining the possible effects of each failure mode on other components of the system that could lead to a hazard(s). For data failures, Lutz and Woodhouse propose the following keywords to guide the analysis to construct a SFMECA table: “incorrect value”, “absent value”, “wrong timing” and “duplicated value”; likewise, for event failures, Lutz and Woodhouse propose the

following keywords to guide the analysis to construct a SFMECA table: “halt/abnormal termination”, “omission”, “incorrect logic/event” and “timing/order” [53].

In [32], Feng and Lutz detail the creation of a product-line SFMEA by using the SFMEA analysis and additionally including an entry to specific which for which product the current failure mode and its effects are being documented. We follow Feng and Lutz [32] in this work by partitioning the SFMECA into separate analyses on the data and events.

However, the work described here differs in that it provides a structured process to create and document a SFMECA table for a multi-agent system product line (MAS-PL) using the Gaia-PL agent-oriented software engineering methodology rather than a general product line.

2.3.2.3 Bi-Directional Safety Analysis

The results of a forward search, such as a Software Failure Modes Effects and Criticality Analysis (SFMECA), and a backward search, such as a Software Fault Tree Analysis (SFTA), will not necessarily be the same, often times both types are utilized in the safety analysis of a safety-critical system [44]. Lutz and Woodhouse developed the Bi-Directional Safety Analysis (BDSA) approach to combine the advantages of the forward and backward search techniques [53]. The forward and backward techniques can be viewed as complementary since the output of the forward technique (i.e., the potential system-wide hazards) should match-up with the inputs of the backward technique. Similarly, the output of the backward technique (i.e., the low-level, local errors that cause a system-wide hazard) should match-up with the inputs of the forward technique [44]. For example, we can verify the completeness of the SFTA by ensuring that every unique hazard listed in the SFMECA table with a particular level of criticality or higher (e.g., major criticality) is a root node within one of the fault trees of the SFTA.

In [32], Feng and Lutz utilize a BDSA to discover incompleteness in the SFMEA and SFTA of a safety-critical product line. In [52], Lutz and Gannod specify a telescope subsystem as a product family and incorporates BDSA to identify additional requirements. Similarly, in [53], Lutz, Helmer, Moseman, Statezni and Tockey performed a forward and backward search for hazards on representative members of a flight instrumentation display product family in hopes of deriving additional safety requirements.

In this work, however, the BDSA is adapted for the use of multi-agent system product lines (MAS-PL) to discover incompleteness in the SFMECA and a Product-Line Software Fault Tree Analysis (PL-SFTA), demonstrate the compliance to safety standards of a MAS-PL by verifying its safety requirements and enable the reuse of safety certification arguments for the MAS-PL.

2.3.3 Summary

Software safety analysis techniques provide software engineers with the tools necessary to identify, analyze and verify the safety requirements of software-critical software systems. Software safety analysis techniques use different approaches to analyze the causes of a system accident and possible hazards of a system failure.

The work described in this dissertation develops product-line safety analysis techniques and tools for safety-critical product lines. In addition, this dissertation develops and integrates these safety analysis techniques and tools for the safety analysis of multi-agent system product lines (MAS-PL).

CHAPTER 3. CASE STUDY: THE PROSPECTING ASTEROID MISSION

The work described in this dissertation is illustrated and evaluated using the Prospecting Asteroid Mission (PAM), a NASA Autonomous Nano-Technology Swarm (ANTS) mission [9], [14], [15], [64], [65], [68], [71], [77], [83], [84]. Like the ANTS-based mission, the PAM spacecraft can be viewed as a multi-agent system product line (MAS-PL) [64], [65], [66]. From a product-line engineering perspective, the similarities in requirements that are to be found on every spacecraft of the PAM swarm (e.g., the navigation and guidance capabilities, the prevention of collisions capability, the ability to warn other spacecraft of an impending solar storm, etc.) can be viewed as product-line commonality requirements.

As described in Chapter 4, the application of a product-line engineering approach to this MAS using our AOSE methodology, Gaia-PL (Gaia – product line) utilizes reusable, core assets to reduce the development effort required. This chapter introduces the ANTS mission as well as the PAM mission to provide necessary context and background information.

3.1 The Autonomous Nano-Technology Swam Mission

The Autonomous Nano-Technology Swarm (ANTS) is a NASA concept mission in the 2015-2030 timeframe that entails a collection of agents that work cooperatively, and autonomously to achieve mission goals [68]. The proposed ANTS technology is a system architecture for scalable, robust and highly distributed systems, and it has been proposed to be used in an family of missions (each with differing objectives and goals), shown in Figure 2, to explore our solar system [9], [14], [15], [64], [68], [71], [77], [83], [84]. For example, the wide range of ANTS-based missions include a swarm of flight-based spacecraft to orbit Saturn and investigate the composition of Saturn's rings [10]

and to travel amongst the asteroid belt between Mars and Jupiter to investigate the composition of asteroids [9], [14], [15], [83], [84] to ground-based spacecraft to look for ice or volcanic material just beneath the surface on Mars [14]. In addition to the NASA-proposed, ANTS-based missions, shown in Figure 2, the United States Department of Defense has shown interest in exploring similar ANTS-based systems using autonomous technologies for the investigation of extreme environments on Earth and for underwater exploration [77].

The ANTS architecture will be based on autonomous, self-addressable, self configuring components that will have the following key aspects [14]:

- Independent, specialized elements
- Multi-level intelligent, autonomous behavior
- Organization via a social insect analogy

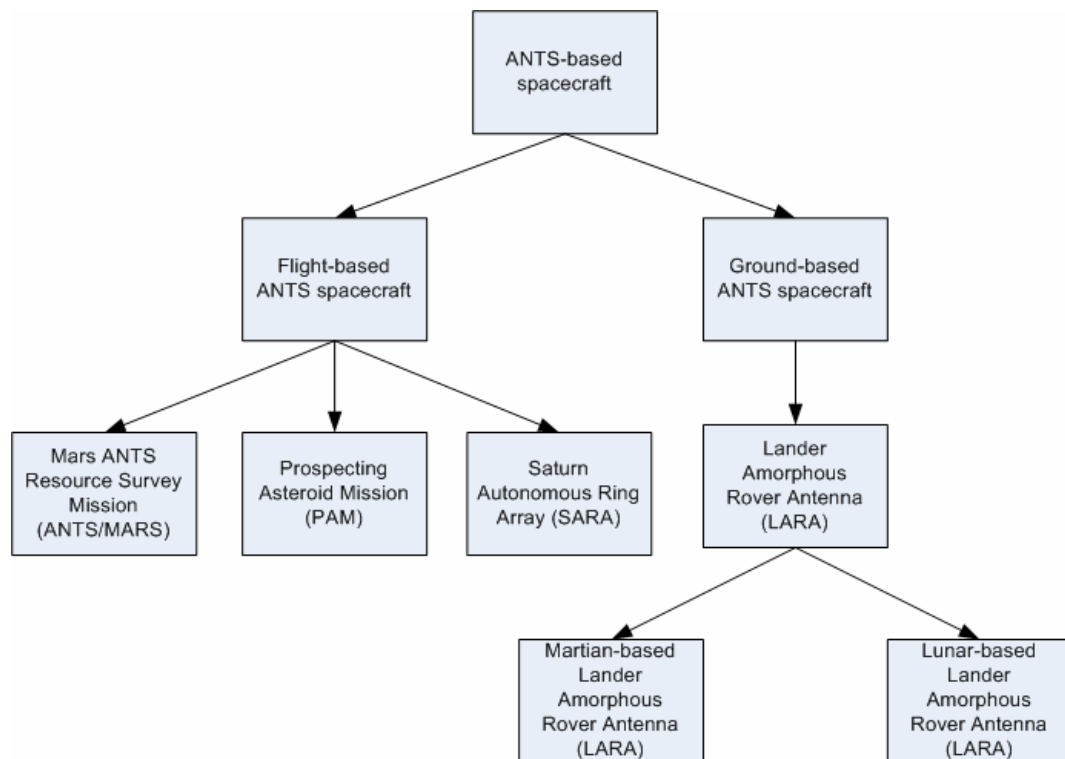


Figure 2 The Family of NASA's Proposed ANTS-Based Missions

Some of the components of the architecture will consist of common subsystems that all spacecraft must have (e.g., inter-spacecraft communication components, guidance and navigation components, etc.) and some components specialized to a small subset of the spacecraft (e.g., X-ray spectrometer components). Thus, the architecture is designed particularly for highly autonomous spacecraft each specialized to perform a specific mission function [14], [15], [68].

The autonomy required by ANTS-based missions will require each spacecraft to have the ability to be self-configuring, self-healing, self-optimizing and self-protecting [77]. Briefly, self-configuring behavior in ANTS-based missions is needed since the nature and objectives of the mission may change as time progresses. For example, new/different science goals may need to be investigated depending on collected data or the current surrounding environment. Self-healing is needed to allow a spacecraft to autonomously discover and recover from malfunctions (e.g., the spacecraft's memory is corrupted as a result from exposure to solar radiation) and be able to continue to perform scientific operations. Self-optimization in ANTS-based spacecraft is desired so that specialized spacecraft (e.g., a spacecraft with a magnetometer) are able to optimize their abilities to perform their scientific objectives and learn how they can better achieve their scientific goals through their learning from the past experiences. Finally, self-protection is needed in each ANTS-based spacecraft so that the spacecraft can prevent itself from harmful situations (e.g., collisions with other spacecraft, radiation from solar storms, etc.).

While these behaviors will have many similarities across all ANTS-based missions (e.g., all ANTS-based spacecraft will be self-protecting by avoiding collisions with other spacecraft), the specific autonomic properties will very depending on the specific mission and the specific objective of a spacecraft (e.g., some spacecraft of the

ANTS-based mission to explore Saturn's rings shall be able to optimize their near-infrared spectrometer to be able to better characterize the ring's composition) [77].

The similarities in the characteristics, behavior and requirements amongst the proposed ANTS-based systems, shown in Figure 2, suggests that adopting a product-line engineering approach may be advantageous in the systems' development since portions of the software engineering assets can be reused across several missions [64], [65], [66]. Using a product-line engineering approach, common components (e.g., the navigation and guidance components and the collision avoidance components of flight-based ANTS systems) can be viewed as product-line commonalities. Similarly, components particular to only some of the ANTS-based spacecraft (e.g., infrared radiometer components for spacecraft specialized to investigate a planet or asteroid's Regolith characterization) can be considered as product-line variabilities. Thus, the family of ANTS-based spacecraft could be built as a multi-agent systems product-line (MAS-PL), as proposed by Peña, Hinchey, Ruiz-Cortés and Trinidad [64], [65], [66].

In this work, we concentrate on a single ANTS-based mission, the Prospecting Asteroid Mission (PAM) [71], [77], [83], [84], as a MAS-PL to illustrate and evaluate our approach. In the following section, the PAM ANTS-based mission is described to provide background and context to the examples and case study presented throughout the remainder of this dissertation.

3.2 The Prospecting Asteroid Mission

The Prospecting Asteroid Mission (PAM) is a currently a 2020-2025 NASA concept mission lasting 5-10 years based on the Autonomous Nano-Technology Swarm (ANTS) technology to explore the asteroid belt between Mars and Jupiter [71], [77], [83], [84]. The proposed PAM mission will consist of up to 1,000 pico-spacecraft (spacecraft weighing less than 1 kilogram) that will autonomously form subswarms to investigate

asteroids of interest in the asteroid belt. In particular, the PAM spacecraft's objective is to search for asteroids that have characteristics indicating that they have resources and material with astrobiologically relevant origins and features. Except for a spacecraft's scientific instrumentation specialties, each PAM spacecraft will have identical hardware.

Each PAM spacecraft will be designated as a *leader* (sometimes called *rulers*), a *messenger* or a *worker* [15], [68], [71], [83], [84]. A spacecraft tasked as a *leader* will determine the types of asteroids and data the mission is interested in and will coordinate the efforts of other spacecraft, in particular *worker* spacecraft, to investigate asteroids to satisfy mission objectives. A spacecraft designated as a *messenger* is tasked with coordinating the communication messages between the *worker* spacecraft, the *leader* spacecraft and the Earth. In addition, the *messenger* spacecraft will, along with the *leader* spacecraft, maintain the position and trajectory data of all spacecraft in the swarm as a requirement for intra-spacecraft communication. *Worker* spacecraft will each contain a single specialized, onboard scientific instrument and be tasked to perform scientific investigation particular its specialized equipment. Types of specialized, onboard scientific instruments that *worker* spacecraft will contain for the PAM mission include spectrometers, altimeters, magnetometers and infrared radiometers.

Currently, there are nine proposed specialized instruments, shown in Table 1, each designated with its own unique objective in the exploration of an asteroid [15], [68], [71], [83], [84]. Of the approximately 1,000 spacecraft proposed for the PAM mission, approximately 80% will be *worker* spacecraft and the remaining 20% will be equally divided amongst the *leader* and *messenger* spacecraft. Thus, for each type of spacecraft there will be a great amount of redundancy since NASA projects that 60%-70% of the PAM spacecraft will be lost over the duration of the mission due to failures, collisions, etc.

Table 1 Types of Specialized Instruments for *Worker* Spacecraft

Worker Specialization	Primary Objective
Visible Imager	Asteroid detection, 3D modeling, Photogeology
Near-Infrared Spectrometer	Mineral abundance mapping
X-ray Spectrometer	Major element/volatile abundance mapping
Gamma-ray Spectrometer	Heavy element/volatile abundance mapping
Neutron Spectrometer	Heavy element/volatile abundance mapping
Altimeter	Shape detection, 3D modeling, Topography, Geomorphology
Radio Science/Magnetometer	Gravity/Magnetic fields mapping, Interior characterization, 3D modeling
Radio Sounder/Infrared Radiometer	Regolith characterization
Neutral Mass Spectrometer	Volatile characterization

To explore the asteroids within the asteroid belt, the PAM spacecraft will autonomously form subswarms of approximately 100 spacecraft, thus forming around 10 subswarms [15], [68], [71], [83], [84]. Each subswarm will spend approximately one month investigating a single asteroid and will consist of several *leader* and *messenger* spacecraft, and the majority of the subswarm will be *worker* spacecraft, of which, several of each instrument specialization will present in every subswarm. The heterogeneous spacecraft of a subswarm will work together to form a “virtual instrument” to investigate an asteroid by combining the data discovered by each of the specialized *worker* spacecraft to form a single model of the asteroid to report to mission control on Earth.

A typical scenario of a PAM subswarm to explore an asteroid within the asteroid belt may be as follows [71]:

The *leader* spacecraft of a subswarm will contain models of the types of science that should be performed. Parts of this model are communicated to *messenger* spacecraft so that the *messenger* spacecraft can relay it to the *worker* spacecraft of the subswarm. Upon receiving a model of what kind of science shall be investigated on an asteroid, the *worker* spacecraft shall take measurements of the asteroid using whatever specialized onboard instruments they have until the collected data of the subswarm's *worker* spacecraft fulfills the model sent by the *leader* spacecraft.

The data will then be sent to a *messenger* spacecraft which will then relay it to the *leader* spacecraft of the subswarm. If this data matches the characteristics that the *leader* spacecraft believe should be further investigated, the *leader* spacecraft will command some *worker* spacecraft equipped with imaging instruments to determine the exact location, size and shape of the asteroid to create a rough model of the asteroid prior to the arrival of other spacecraft so that they can have a model for maneuvering around the asteroid and avoid collisions. Other spacecraft would then work together to finish the model and mapping of the asteroid. This process is partially illustrated in Figure 3.

From this scenario, the PAM spacecraft can be viewed hierarchically as acting as a team of teams where some teams last longer than others since the scientific capabilities of spacecraft differ and because each team is temporarily dedicated to a specific task or objective [15], [68], [71], [83], [84].

In terms of safety, an ANTS requirement, which must hold for the PAM mission, calls for **no single-point failures** [15], [68], [71], [83], [84]. This implies that there must not be a single, central *leader* spacecraft that commands the entire swarm, thus the need for high redundancy in the spacecraft and in the swarm to ensure, for example, that there is not a single *leader* spacecraft responsible for commanding the entire swarm.

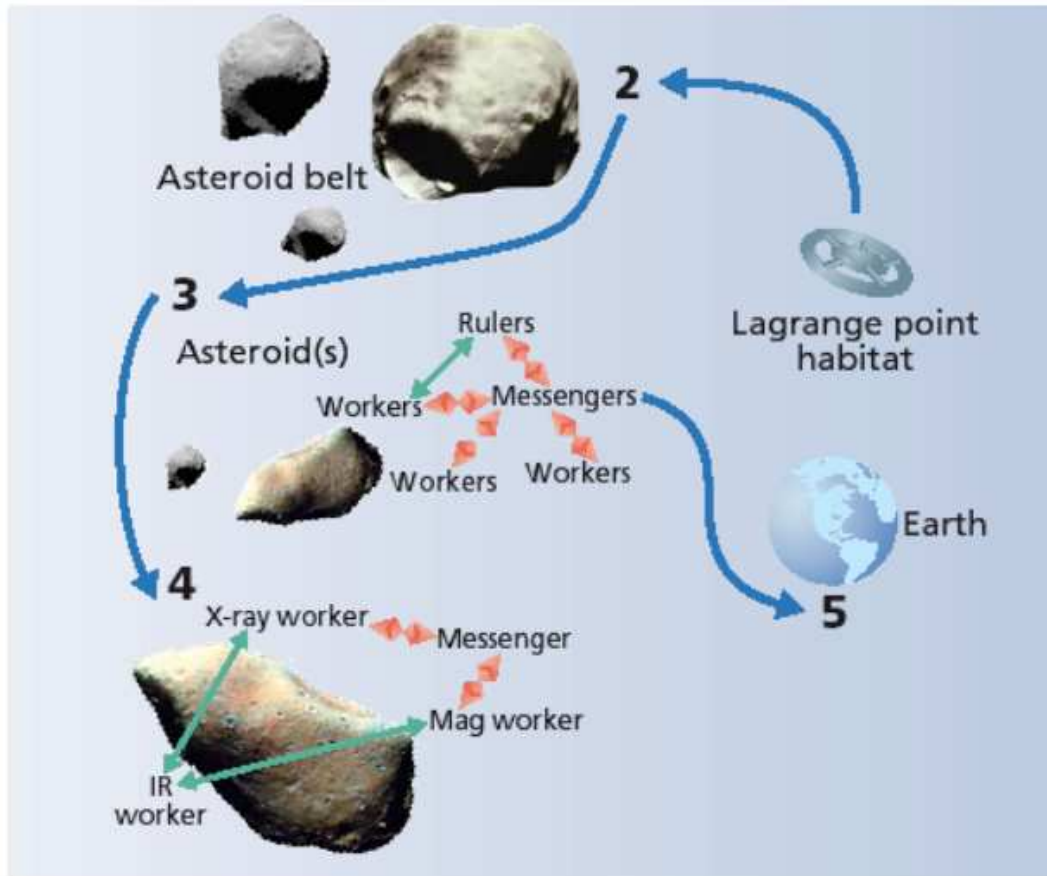


Figure 3 PAM Spacecraft Exploring the Asteroid Belt (from [77])

Additionally, the extreme conditions of space will necessitate other safety-critical requirements to be placed on PAM spacecraft to avoid hazardous situations [15], [68], [71], [83], [84]. One such situation is that the PAM spacecraft will need to protect themselves from the solar radiation present during a solar storm. To protect the swarm from solar radiation, some spacecraft will be tasked, in addition to their other objectives, to monitor the solar disc for an impending solar storm. To meet the no-single point failure requirement, several spacecraft will be tasked with monitoring the solar disc for an impending solar storm.

However, not all of these spacecraft will be actively monitoring the solar disc although they all have the capability. Rather, some spacecraft will switch from not monitoring the solar disc to actively monitoring the solar disc when it is determined that the swarm requires additional monitoring spacecraft (e.g., when previous spacecraft monitoring the solar disc have been lost due to failure, collision, etc.). When a solar storm is detected, this spacecraft will warn the entire swarm to take protective measures. A PAM spacecraft receiving such a warning will relay this message to other nearby spacecraft and may power down its subsystems and use its solar sail as a shield to protect itself from the harmful effects of solar radiation.

To preserve mission-critical requirements (i.e., the swarm's ability to pursue scientific goals and report their findings), additional capabilities will be given to some spacecraft of the PAM swarm to achieve redundancy at the swarm level. Similar to the ability that some spacecraft will go from not monitoring the solar disc for impending solar storms to actively monitoring it, some spacecraft may be able to switch from a *leader* spacecraft to a *messenger* spacecraft or vice versa if conditions of the swarm determine that additional *messenger* spacecraft or *leader* spacecraft, respectfully, are needed to due the loss or failure of spacecraft [15], [68], [71], [83], [84].

Like the ANTS-based mission, the PAM spacecraft can be viewed as a multi-agent system product line (MAS-PL) [64], [65], [66]. From a product-line engineering perspective, the similarities in requirements that are to be found on every spacecraft of the PAM swarm (e.g., the navigation and guidance capabilities, the prevention of collisions capability, the ability to warn other spacecraft of an impending solar storm, etc.) can be viewed as product-line commonality requirements. Similarly, the differences amongst the spacecraft of the PAM swarm (e.g., the differing requirements between *leader*, *messenger* and *worker* spacecraft, the ability of some spacecraft to monitor the solar disc for impending solar storms, the ability of some *messenger* spacecraft to be

upgraded to a *leader* spacecraft, etc.) can be considered as product-line variability requirements.

This work uses NASA's ANTS-based PAM mission as a case study throughout the remainder of this dissertation to motivate, illustrate and evaluate our Agent-Oriented Software Engineering (AOSE) methodology, Gaia-PL (Gaia - Product Line), to construct a MAS-PL so that its software engineering assets can be reused when building new systems. Chapter 4 describes the reduction in the development effort required as a result of the application of a product-line engineering approach using Gaia-PL to this MAS.

Further, we use the PAM mission described in this chapter to motivate, illustrate and evaluate our product-line safety analysis techniques ability to evaluate and improve the safety of a MAS-PL in a way that the produced safety analysis assets are reusable for future systems. Chapter 5 details the product-line safety analysis techniques and tools ability to identify, analyze and verify the safety requirements of the PAM mission. In addition, we evaluate safety analyses value as producing reusable safety artifacts and their ability to reduce development costs compared to a non-product line safety analysis approach.

CHAPTER 4. DEVELOPING MULTI-AGENT SYSTEM PRODUCT LINES USING THE GAIA-PL METHODOLOGY¹

Chapter 1 stated as a thesis that an Agent-Oriented Software Engineering (AOSE) methodology can be devised to enhance the reuse in the design and development of a safety-critical, multi-agent system (MAS) by incorporating software product-line engineering principles to develop reusable software engineering assets in a way that allows software engineers to take advantage of the reusable assets to create a MAS. Based on the foundation of background information and related research given in Chapters 1 and 2, this chapter describes our Gaia-PL (Gaia – Product Line) AOSE methodology to design and develop multi-agent system product lines (MAS-PL)² using software product-line principles. This chapter details how to develop reusable requirement specifications for a MAS-PL and then reuse them for initial system development as well as during evolution. To illustrate and evaluate our Gaia-PL methodology, we use the Prospecting Asteroid Mission (PAM) case study described in Chapter 3.

¹ This chapter extends our previous work that has appeared in papers at the *2005 International Conference on Software Engineering Workshop on Software Engineering for Large-Scale, Multi-Agent Systems (SELMAS'05)*, a 2006 chapter in *Software Engineering for Multi-Agent Systems IV, Lecture Notes In Computer Science*, co-authored with Robyn R. Lutz as well as a forthcoming book chapter entitled *Current Research in Multi-Agent System Product Lines (MAS-PL)*, co-authored with Joaquin Peña, Antonio Ruiz-Cortes, Michael Hinchey and Robyn R. Lutz.

² The term multi-agent system product line (MAS-PL) was coined by Joaquin Peña, Michael Hinchey, Antonio Ruiz-Cortes and Pablo Trinidad for what we previously called product-line, multi-agent systems (PL-MAS).

4.1 Integrating Software Product-Line Engineering Principles into the Gaia Methodology

This section examines the need for the integration of software product-line engineering principles into the design and development of MAS and describes our approach of using an agent's variation points as a mechanism to include software product-line engineering principles into our Gaia-PL methodology.

4.1.1 The Need for Reuse in Developing Multi-Agent Systems

Reuse is highly desirable in software engineering as a way to reduce the cost of the design and development of software. Software reuse technologies have been a driving force in significantly reducing both the time and cost of software requirements specification, development, maintenance and evolution. Industry's continuous demand for shorter software development cycles and lower software costs encourages software development methodologies to exploit software reuse principles whenever possible.

Agent-Oriented Software Engineering (AOSE) methodologies have provided software engineers with the mechanisms to understand, model and develop complex multi-agent systems (MAS). From its onset, one of the goals of AOSE has been to provide methodologies for reusing and maintaining agent-based software systems [85]. Despite this, no methodology has provided software engineers with the reuse mechanisms at an early stage in the software development life cycle (i.e., requirement specification phase). The realization of MAS development partially depends upon whether AOSE can achieve reductions in development time and cost comparable to other reuse-conscious software development methods [7].

Software product-line engineering, discussed in Chapter 2, is one such reuse technology that supports the systematic development of a set of similar software systems by understanding, controlling and managing their common, core characteristics and their

differing variation points [12], [67]. The benefits of the product-line concept come from the reuse of the common requirements of the product line in the development of a new product-line member [76]. Thus, the assets gained from the initial engineering of the product line can be at least partially applied to any new product-line member.

The Gaia-PL methodology provides a requirements specification pattern to capture the dynamically changing design configurations of agents and reuse the requirement specifications for future similar systems. This is achieved by adopting a product-line approach into AOSE by capturing the dynamically changing design configurations of agents as product-line variation points and reusing them for future systems. The use of variation points in Gaia-PL for the design and development of MAS is discussed in the following section.

4.1.2 Using Variation Points in Multi-Agent Systems

The Gaia methodology centers on defining an agent based upon the role(s) that it can assume during its lifetime [92], [94]. Each role's requirements specification is defined by its protocols (i.e., defines how agents interact), activities (i.e., the computations associated with the role that can be executed without interacting with other agents), permissions (i.e., the information resources that the role can read, change and generate) and responsibilities (i.e., the liveness and safety properties the role must ensure).

However, Gaia has three limitations of interest to the work presented in this dissertation. First, although Gaia provides a mechanism to allow the role of an agent to change dynamically, it is unclear how to document agent requirements specifications during the analysis and design phases when an agent must be updated to include new functionality. Second, the design of an agent in Gaia is not hierarchical [42]. That is, the roles of an agent are coarsely defined allowing little flexibility (i.e., little opportunity for

reuse) for similar, yet slightly different behavior in the same role in different agents. Third, the Gaia methodology fails to provide a mechanism by which the requirements specification templates developed during the analysis phase can be reused to be incorporated into the current system or to build a new, similar but slightly different system.

Gaia-PL addresses these limitations by introducing *variation points* into the design and development of MAS. Product-line engineering uses variation points to capture the allowed differences amongst members belonging to the same product family. For Gaia-PL, we define the variation points for a specific role of an agent as the differing protocols, activities, permissions and responsibilities available to that role. Variation points typically stem from the grouping of the product-line variabilities defined in the Commonality and Variability Analysis (CVA), discussed in Section 2.1.1, documented as part of the output of the Requirements Documentation phase of Gaia-PL, discussed in the next section.

The introduction of variation points in Gaia-PL addresses the limitations of Gaia by allowing the software engineer to define a role with greater flexibility and partition some functionality of a role depending on the agent and system's current configuration. The variation point notion is important because it aids in capturing the different arrangements of agents and promotes reuse.

4.1.2.1 Variation Points

Variation points are added with the Gaia characteristics of a role [92], [94]. This allows Gaia-PL to leverage a product-line-like perspective to maximize reuse among software products that share a great many similarities amongst each other and differ by only a few variations. In the following paragraphs, examples of variation points are given to illustrate this.

From previous work [19] [21], we have shown that an important way to classify variation points for an agent of a MAS is based on the varying intelligence levels for a specific role. For example, in the TechSat21 satellite constellation [8], [73], a cancelled NASA-proposed, agent-based, satellite constellation comparable to the Prospecting Asteroid Mission (PAM) case study used in this dissertation, variation points for a role were ordered in terms of increasing intelligence levels, I4 through I1, defined as follows:

- I4: the role is able to receive and execute commands
- I3: the role is able to participate in local planning activities pertinent to the role as well as receive and execute commands
- I2: the role is able participate in local planning and interaction activities pertinent to the role, contains partial cluster-knowledge related to the role's objective as well as receive and execute commands
- I1: the role is able participate in cluster-level planning and interaction activities pertinent to the role, contains full cluster-knowledge related to the role's objective as well as receive and execute commands

Thus, in this example, as a role in a TechSAT21 satellite is promoted to a higher intelligence level (from I3 to I2, for example) the configuration of the agent dynamically changes by incorporating additional protocols, activities, permissions and/or responsibilities. The reverse occurs when a role is demoted from a higher intelligence level to a lower intelligence level (from I2 to I3, for example). Using this construct, our notion of *an agent's role may have one or more variation points*.

The actual decision as to which features to group together and how to classify each variation point is domain and/or application specific and is not covered in this work. Rather, we assume that domain experts group the variabilities listed into variation points so that they can be used during the analysis phase of Gaia-PL. However, in the application of Gaia-PL to the PAM case study, we found that the variation points were

intuitively identifiable from the functionality described in the variability requirements of the differing spacecraft.

The variation points will initially be fixed upon deployment of the MAS based upon the software and hardware facilities available to the agent as well as the role's goal. At deployment a default variation point for each role is set. During execution, a role may change its variation point (e.g., intelligence level) based upon its internal state, commands from external sources or the environment.

Alternatively, within a distributed, agent-based system, it is not likely that the same set of variation points will be included in any given role throughout the entire MAS [19]. Thus, from a product-line engineering perspective, we can view the set of roles containing different role/variation point combinations as a product line. The set of roles and dynamic variation points an agent contains is its *configuration*.

For example, in a small case study on the application of an earlier version of Gaia-PL to the TechSAT21 case study we performed in [19] [21], the intelligence levels listed above describe the variation points for a role that was tasked to perform allocation planning for the TechSAT21 satellites to equalize the fuel use across the entire cluster. Any agent with this role would be assigned a variation point based on the intelligence level, I4-I1, it is capable of assuming during its lifetime. One agent may be assigned an I4 intelligence level for this variation point. This implies that this specific agent can never increase its intelligence level (i.e., be upgraded) any higher. However, an agent assigned with an I2 intelligence level for this role's variation point has the configuration so that, at any point in its lifetime, it may be operating at the I4, I3 or I2 intelligence level. This may be useful for systems that require redundancy. For example, the agent assigned with an I2 intelligence level for this role's variation point may primarily operate at the I3 intelligence level and only be upgraded to the I2 intelligence level if it is needed to

assume the planning for another agent operating at the I2 level that is failing, has been damaged or needs replacement.

The intelligence level variation point of this example will not be universal to all agent-based, distributed systems. Variation points are particular to each application and, indeed, particular to each role. For example, other variation points could include active, passive; hot-spare, cold-spare; etc.

For the PAM case study used in this dissertation, several different types of variation points were identified for the various roles of the spacecraft (i.e., agent). Note that for the PAM case study, we define an *agent* at the spacecraft-level. This follows other work on PAM by Peña, Hinchey, Ruiz-Cortés and Trinidad in [63] [64]. This additionally follows our previous work in applying Gaia-PL to the TechSAT21 case study in [19] [21] and other work by Das, Krikorian and Truszkowski in [16] and Schetter, Campbell and Surka in [73].

In the PAM case study, one of the important variation points we identified for the roles of an agent as based on whether the spacecraft was to be a *leader*, *messenger* or *worker* spacecraft for the PAM swarm. For some roles that we identified in the PAM case study, further described in Section 4.2, functionality will slightly differ depending on what kind of spacecraft it is (i.e., *leader*, *messenger* or *worker*). However, despite the slight differences in functionality, a majority of the functionality will be common regardless of what kind of spacecraft it is. For example, each PAM spacecraft will have a *Self-Optimizer* role that is tasked with improving its ability to identify, explore and communicate data of an asteroid. While some functionality of this role will be common to all types of PAM spacecraft (e.g., the ability check the spacecraft's current power consumption, check the status of the solar sails, calculate the spacecraft's position and current velocity, etc.), other functionality of the *Self-Optimizer* role will be tailored to the type of spacecraft. For example, a *leader* spacecraft will additionally require functionality

continuously optimize its ability to decide what kinds of asteroid to investigate past on recent historical data. Further, a *worker* spacecraft will additionally require functionality to be able to optimize the use of its onboard, specialized scientific instrument via repositioning itself, altering its scientific goal, etc. Finally, a *messenger* spacecraft will additionally require functionality in the *Self-Optimizer* role to be able to optimize its facilitation of the swarm’s communication network by deciding what messages should be sent to other spacecraft, repositioning itself to best communicate with other spacecraft, etc. In this case, all PAM spacecraft will share the common functionalities for the *Self-Optimizer* role and will then be further specialized with its appropriate, extended *Self-Optimizer* role variation point depending on what type of spacecraft it is. Note that requirement specifications for the *Self-Optimizer* role, and all other roles of the PAM case study, can be found in the Gaia-PL Role Schemas listed in Appendix D.

Besides defining the variation points of a role for a PAM spacecraft based on the type of spacecraft that it is (i.e., *leader*, *messenger* or *worker*), we found that other variation points could be defined for other roles. For example, a *leader* spacecraft of the PAM swarm will have a role called *LeaderPlanner* that is tasked with managing, planning and coordinating the spacecraft of a PAM subswarm so that the subswarm can effectively pursue and satisfy system-wide and individual scientific goals. For this role, we identified the variation points as follows:

- **Passive:** Acts as a backup to verify/double-check the commands and calculations of a spacecraft with a *LeaderPlanner* role acting with the “active” variation point; does not actually command spacecraft, only calculates, verifies the actions to be performed
- **Active:** Able to command the spacecraft of a PAM swarm regarding its plan to coordinate that spacecraft regarding their pursuit of scientific

goals; request from “passive” *LeaderPlanners*
verification/agreement on its calculated strategy

In this role, a *Leader* spacecraft’s *LeaderPlanner* role will be configured as either passive only or both passive and active. Again, a *LeaderPlanner* role configured with both the passive and active variation points may only assume one of the variation points at a time. This may be useful in the event that a *Leader* spacecraft acting as a backup (i.e., the spacecraft’s *LeaderPlanner* role acting at the “passive” variation point although it is also capable of the “active” variation point) needs to assume an “active” *LeaderPlanner* role if another *Leader* has failed.

However, not every role that can be defined for an agent will necessarily have variation points. For those roles that have no variations amongst the agents of a MAS, no variation points should be defined. This implies that, for any agent with a role that has no defined variation points, the functionality will be identical. To accommodate this, Gaia-PL does not require defined variation points for every role and, rather, follows the Gaia approach for those roles without identified variation points.

In the PAM case study, for example, one such role was the *Navigator* role. This role is tasked with providing the PAM spacecraft with the functionality to maneuver itself in space using its solar sail. This functionality is required in all PAM spacecraft identically regardless of the type of spacecraft (i.e., *leader*, *messenger* or *worker*) or any other possible variations.

4.1.2.2 Binding Time in Variation Points

For every variation point identified, a *binding time* is associated to it which defines the time at which the variation point could be assumed by a role. Potential binding times include design-time, specification-time, configuration-time and run-time.

In the case of our PAM case study, most of the binding times were at design-time. For example, the *Self-Optimizer* role's variation points of *Leader*, *Messenger* or *Worker* must be decided for a specific PAM spacecraft while it is being designed. Thus, designers would have to integrate the functionality associated with the chosen variation point with the common functionality to the *Self-Optimizer* role found in all spacecraft.

For the *LeaderPlanner* role, however, the binding time is not straight forward. The decision for whether a spacecraft with the *LeaderPlanner* role should have only the "passive" variation point or both the "passive" and "active" variation point must be done at design time. Yet, for those *LeaderPlanner* roles that have both the "passive" and "active" variation points, the ability to switch from "passive" to "active" or vice versa, based on its own decision or on a command received, is done at runtime. Thus, the decision for the possible configurations of this variation point is decided upon at design-time, the ability for the spacecraft to alter its configuration for this variation point is at runtime.

In the application of the Gaia-PL methodology to the PAM case study, we found that the binding time of a role's variation point often followed that of the *LeaderPlanner* example described above. This is likely a core characteristic of many MAS because of their need to be autonomous and adapt to the changing situation and environment. For example, the need for the *LeaderPlanner* role to be either "passive" or "active" is primarily due to the need for the PAM swarm to be highly redundant and able to reconfigure itself in the event of failure.

Identifying the variation points to which a role may dynamically switch, such as shown in the *LeaderPlanner* role, allows us to classify at which variation points the protocols, activities, permissions and/or responsibilities are introduced to the role. Partitioning the requirements specifications (i.e., the protocols, activities, permissions and responsibilities) of an agent in this manner will allow us to reuse the requirement

specifications for future systems. Thus, future agents within a domain such as Earth-orbiting microsattellites can more readily utilize assets that have been specified in such a way. These future systems employ roles comprising some of the variation points previously defined as well as new capabilities not found in any of the previous systems.

4.1.2.3 Gaia and Variation Points

In the beginning of this section, Section 4.1.2, it was stated that the Gaia methodology [92], [94] has the following limitations:

1. It is unclear how to document an agent's requirements specifications during the analysis and design phases when an agent must be updated to include new functionality particularly when the role of an agent can change dynamically
2. The roles of an agent are coarsely defined allowing little flexibility for similar, yet slightly different behavior in the same role in different agents because the design of an agent's roles in Gaia is flat rather than hierarchical [42]
3. There is no clear mechanism by which the requirements specification templates developed during the analysis phase can be reused to be incorporated into the current system or to build a new, similar but slightly different agent

The ability to define and document variation points in Gaia-PL specifically addresses these limitations in Gaia to facilitate the reuse of the requirement specifications for several, similar but slightly different agents.

For agents that have roles that may dynamically change its functionality during its lifetime, the ability to partition a role's varying functionality via its variation points allows the designer to specify the possible configurations of the role (i.e., the selection of the variation points that the role may assume during its lifetime) at an early binding time (i.e., design-time, specification-time). Then, that particular role can assume (or be

commanded by another spacecraft to assume) a particular variation point of the role during runtime. The *LeaderPlanner* role of the PAM case study described in Section 4.1.2.2 illustrated this situation. Thus, the variation points provide a mechanism to capture the functionality of a role that may dynamically change during execution. The mechanism to document the roles, variation points and binding times for the agents of a MAS-PL is detailed and illustrated in Section 4.2.

Partitioning the role of an agent into its common parts and its variable parts (i.e., the variation points), Gaia-PL provides software engineers with the ability to define a role hierarchically. Using this approach, the common functionality of a role is captured and the variable functionality is captured as the variation points at a level below. The use of a Feature Model aids in structuring the roles and variation points of an agent hierarchically. This is further detailed and illustrated in Section 4.2.1.

Structuring the roles and variation points of an agent in a hierarchical manner and partitioning the common and variable functionality of a role allows for flexibility and reuse of the requirements specifications of the roles and variation points. These requirements specifications can be reused for similar, yet slightly different agents during the initial development of a MAS as well as during evolution. This is further detailed and illustrated in Section 4.3 and contrasted with the Gaia methodology in Section 4.4.

4.2 Documenting the Requirements Specifications of a MAS-PL in the Gaia-PL Methodology

This section describes the Requirements Documentation, Analysis and Design, and the Detailed Design phases of the Gaia-PL methodology. The process and software engineering artifacts generated from these phases are illustrated in Figure 4. This figure illustrates the Gaia-PL methodology in context to the phases of Gaia (i.e., Requirements Documentation, Analysis and Design, and the Detailed Design) and Weiss and Lai's

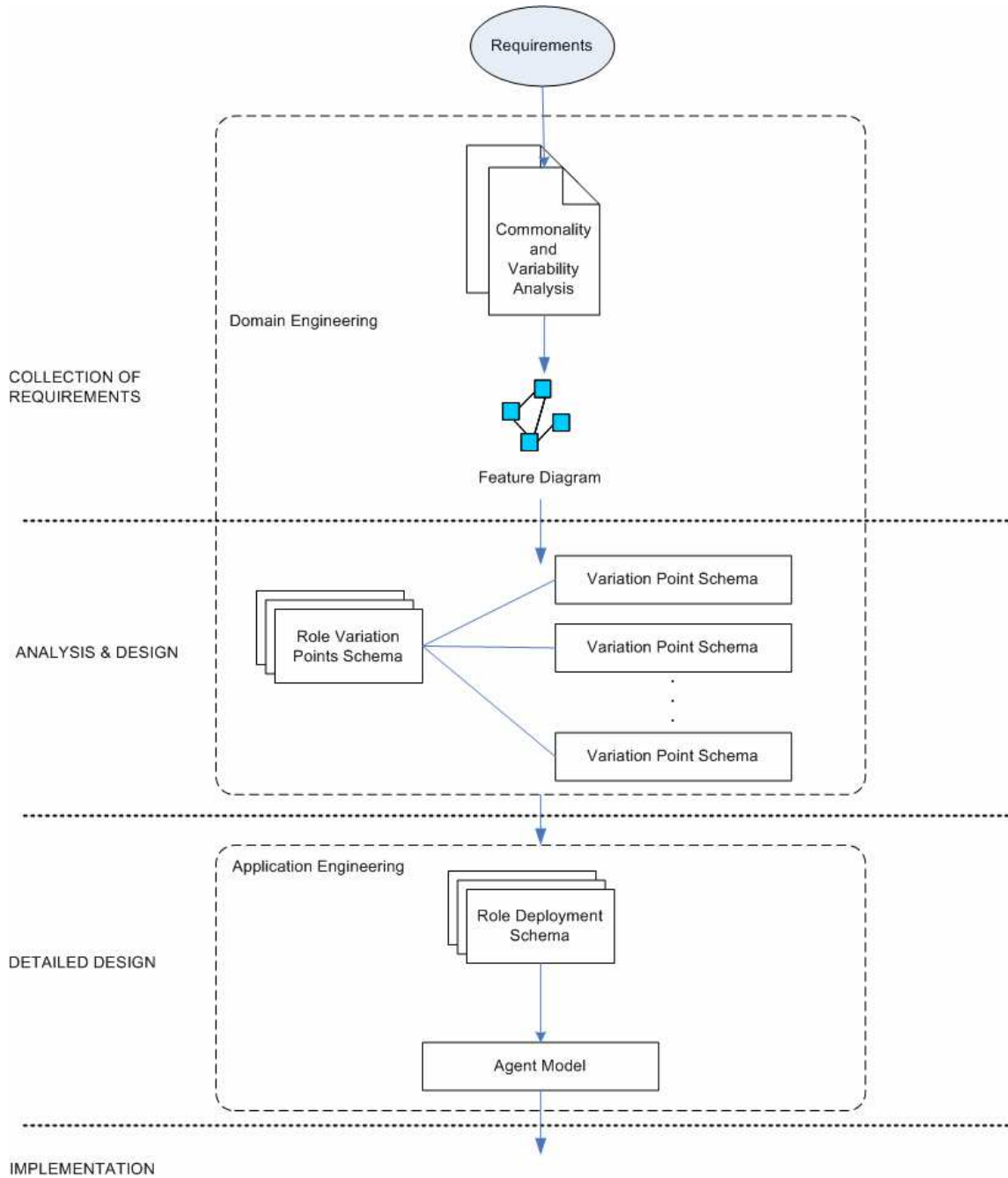


Figure 4 An Overview of the Software Engineering Artifacts of Gaia-PL

Family-Oriented Abstraction, Specification and Translation (FAST) [88] product-line engineering approach. For each phase, we describe the documentation process and how each document will later contribute to the ease of reuse, discussed in Section 4.3.

Although Gaia-PL is detailed as its own methodology in this chapter to develop and document the requirements and requirements specifications of a multi-agent system product line (MAS-PL), Gaia-PL can be applied as an extension to the Gaia methodology [92], [94], shown in Figure 1. This would entail using the Gaia-PL schemas and procedure discussed in this chapter for the requirements specifications and other Gaia methodology's models and schemas for other parts of an agent-oriented system.

4.2.1 Requirements Documentation Phase

The Requirements Documentation phase of the Gaia-PL methodology involves identifying and documenting the multi-agent system product line's (MAS-PL) commonality and variability requirements. This section describes the Commonality and Variability Analysis (CVA), the Parameters of Variation Table(s), the Feature Model and the use of DECIMAL [23], [58], [59] to facilitate the requirements documentation process.

4.2.1.1 The Commonality and Variability Analysis

Documenting the requirements of a multi-agent system product line (MAS-PL) in Gaia-PL follows the same principles of software product-line engineering. In the development of a software product line, requirements are collected and then documented in a Commonality and Variability Analysis (CVA) as well as a Parameters of Variation table for the variability requirements [1], [70], [88]. The requirements engineering process of [1], [70], [88] to gather, identify and document the product-line requirements in a CVA for a product line can be used in Gaia-PL and is thus not covered here.

Alternative approaches to the CVA in documenting product-line requirements and performing variability analysis include the goal-oriented [5] or the feature-oriented [43] approach. Alternatively, the use of domain or application expertise may also suffice in this process. This work exclusively used the CVA as the medium for variability documentation and analysis because of our use of the FAST methodology (in which a CVA is exclusively utilized to document and analyze variabilities). In terms of reuse, CVA is superior to either goal-oriented or feature-oriented approaches since it clearly defines those requirements that will be found in every member of a product line (i.e., commonalities) and those requirements that will only be found in a subset of the members of a product line (i.e., variabilities).

In the PAM case study, we identified a total of 35 high-level commonality requirements and 62 variability requirements to document in the CVA. Excerpts from the CVA for the PAM MAS-PL are shown in Figure 5 (Commonalities) and Figure 6 (Variabilities). The entire CVA for the PAM case study can be found in Appendix A.

From the CVA's variabilities, the Parameters of Variation table can be derived to better define the variabilities listed in the CVA [88]. The Parameters of Variation tables listed the parameters name, the associated variability requirement (for traceability), a description of the parameter, the domain of the possible values of the parameter, the binding time at which the configuration of the parameter must be selected.

In the PAM case study, 48 parameters of variation were found from the 62 variability requirements. Note that several product-line variabilities can constitute a single parameter of variation. For example, for the variability "A spacecraft performing subswarm allocation and planning may vary in its role in allocation and planning activities", the domain of parameter values for this variability is [passive, active]. Note that variability corresponds to the *LeaderPlanner* role discussed in Section 4.1.2.1.

COMMONALITIES

General Commonality Requirements

- C_G1. The PAM swarm shall have no single point of failure [15].
- C_G2. The PAM swarm shall be robust to minor faults and catastrophic failures [14].

Self-Optimization Commonality Requirements

- C_SO1. Every spacecraft shall be able to adjust to the surrounding environment [77], [84].
- C_SO2. Every spacecraft shall be able to optimize itself through calibrating its instruments [77], [83], [84].
- C_SO3. Every spacecraft shall be able to optimize its power consumption [15], [65], [66], [84].
- C_SO4. Every spacecraft shall be able to monitor and adjust its relative positions to optimize its scientific exploration [77], [84].

Self-Healing Commonality Requirements

- C_SH1. Every spacecraft shall be able to recognize that its memory is corrupted/damaged [64], [65], [66], [84].
- C_SH2. Every spacecraft shall be able to request an uncorrupted memory from another spacecraft in the event that it recognizes that its memory is corrupted [71], [84].
- C_SH3. Every spacecraft shall be able to send its uncorrupted memory to another spacecraft upon request [71], [84].

Self-Protection Commonality Requirements

- C_SP2. Every spacecraft shall be able to communicate with nearby spacecraft in order to prevent collisions [64], [66], [71], [77], [84].
- C_SP3. Every spacecraft shall be responsible for preventing collisions with asteroids [64], [65], [66], [71], [77], [84].
- C_SP4. Every spacecraft shall be able to store a 3D map of nearby asteroids in order to prevent collisions [71], [77], [84].
- C_SP5. Every spacecraft shall be able to take acceptable risks while attempting to satisfy its scientific goals [71], [77], [84].
- C_SP6. Every spacecraft shall be able to deploy its solar sail to use as a shield for protection against solar storms [66], [77], [83], [84].
- C_SP7. Every spacecraft shall be able to switch off its subsystems when needed to protect against solar radiation [66], [77], [83], [84].
- C_SP8. Every spacecraft shall be able to receive messages from other spacecraft giving advanced warning of an impending solar storm [65], [66], [77], [84].

Miscellaneous Commonality Requirements

- C_M1. Every spacecraft shall have the ability to control its own guidance navigation and control functions [14], [15], [83].
- C_M3. Every spacecraft shall be able to use their solar shields as its means of flight [14], [15], [65], [66].

Figure 5 Excerpt of the Commonalities from the Commonality and Variability Analysis for the PAM MAS-PL

VARIABILITIES

Self-Optimization Variability Requirements

- V_SO1. A spacecraft's ability to optimize itself via improving their ability to identify asteroids of interest may vary [15], [71], [77], [83] [84].
- V_SO2. A spacecraft's ability to share its optimization information regarding the identification of asteroids of interest with *leader* spacecraft may vary [77], [84].
- V_SO3. A spacecraft's ability to optimize itself through positioning itself appropriately to best facilitate communications with *messenger* spacecraft may vary [15], [77], [84].
- V_SO4. A spacecraft's ability to share its optimization information regarding positioning itself appropriately to best facilitate communications with *messenger* spacecraft may vary [15], [77].
- V_SO5. A spacecraft's ability to optimize itself via learning through their past experiences to better investigate an asteroid may vary [15], [77], [84].
- V_SO6. A spacecraft's ability to share its optimization information regarding how to better investigate an asteroid with *worker* spacecraft may vary [15], [77], [84].

Self-Protection Variability Requirements

- V_SP1. A spacecraft's ability to be tasked with constantly observing the solar disc to detect signs of an impending solar storm may vary [65], [66], [77], [84].
- V_SP2. A spacecraft's ability to receive warnings from mission control of an impending solar storm may vary [65], [66], [77], [84].

Leader Spacecraft Variability Requirements

- V_L1. A spacecraft's ability to be in charge of performing subswarm allocation and planning may vary [15], [71], [83], [84].
- V_L2. A spacecraft performing subswarm allocation and planning may vary in its role in allocation and planning activities [15].
- V_L3. A spacecraft's ability to be able to assign teams of *worker* and *messenger* spacecraft may vary [83].
- V_L4. A spacecraft's ability to direct/coordinate *worker* spacecraft to investigate a specific asteroid may vary [77], [83], [84].
- V_L6. A spacecraft's ability to be responsible for determining the types of asteroids to investigate may vary [71], [77], [83], [84].

Messenger Spacecraft Variability Requirements

- V_M1. A spacecraft's ability to relay/coordinate messages between *worker* spacecraft and *leader* spacecraft may vary [15], [71], [77].
- V_M2. A spacecraft's ability to relay/coordinate messages between *leader* spacecraft and mission control may vary [15] [71], [77].

Figure 6 Excerpt of the Variabilities from the Commonality and Variability Analysis for the PAM MAS-PL

Table 2 Excerpt of the Parameters of Variation Table for the PAM MAS-PL

Parameter	Meaning	Domain	Binding Time	Default
GENERAL VARIABILITY REQUIREMENTS				
P1: vSpacecraftRole V_G1	The role that a spacecraft is to initially assume.	[Leader, Messenger, Worker]	Design	Worker
SELF-OPTIMIZATION VARIABILITY REQUIREMENTS				
P4: vIdAsteroidsOptimization V_SO1, V_SO2	The ability of a <i>leader</i> spacecraft to optimize its ability to identify asteroids of interest and share this information with other <i>leader</i> spacecraft.	[True, False]	Specification	False
P5: vCommOptimization V_SO3, V_SO4	The ability of a spacecraft to optimize its positioning for communications and sharing this optimization with other spacecraft.	[True, False]	Specification	True
P6: vScienceOptimization V_SO5, V_SO6	The ability to optimize its scientific exploration of an asteroid and sharing this optimization with other spacecraft.	[True, False]	Specification	False
SELF-PROTECTION VARIABILITY REQUIREMENTS				
P7: vSolarDiscWatch V_SP1	The ability of a spacecraft to constantly watch the solar disc for the signs of an impending solar storm.	[Passive, Warm-Spare, Active]	Design	Passive
P8: vMissConStormWarn V_SP2	The ability of a spacecraft to receive messages from mission control warning of an impending solar storm.	[True, False]	Design	False
MESSENGER SPACECRAFT VARIABILITY REQUIREMENTS				
P20: vRelayMessagesSwarm V_M1, V_M4	The ability to relay and coordinate messages between spacecraft.	[True, False]	Specification	False
P21: vRelayMessagesMisCon V_M2	The ability to relay and coordinate messages to mission control.	[True, False]	Specification	False

Similarly, for the variability “A spacecraft’s ability to be tasked with constantly observing the solar disc to detect signs of an impending solar storm may vary” the domain of parameter values for this variability is [passive, warm-spare, active]. An excerpt of the Parameters of Variation table, including these two examples, is shown in Table 2. The entire Parameters of Variation Tables are given in Appendix B. Note that the CVA shown in Figure 5 and Figure 6 and the Parameters of Variation Table shown in Table 2 will also be used in Chapter 5 to illustrate the safety analysis of a MAS-PL.

4.2.1.2 Using DECIMAL to Document the Requirements

To document the Commonality and Variability Analysis (CVA) of the, we utilize the DECIMAL tool [23], [58], [59], shown in Figure 7. Within DECIMAL, the commonalities and the variabilities, and their associated parameters of variation can be documented. For example, the variability V_SP1 (from the CVA in Figure 6): “A spacecraft’s ability to be tasked with constantly observing the solar disc to detect signs of an impending solar storm may vary” with parameter of variation of [passive, warm-spare, active] (from the Parameters of Variation Table in Table 2) is shown in Figure 8.

Although DECIMAL only provides a digital medium in which to document the commonality and variability requirements of a multi-agent system product line (MAS-PL) in Gaia-PL, we use it for two reasons. First, DECIMAL provides a convenient mechanism to document and store the requirements of a MAS-PL during Gaia-PL’s Requirements Documentation Phase as well as providing an automated check to verify that an agent’s variable requirements abide by the MAS-PL’s dependencies during Gaia-PL’s Detailed Design Phase, discussed in Section 4.2.3. Second, for safety-critical MAS-PL, DECIMAL is used in conjunction with the safety analysis techniques and tools we describe in Chapter 5.

4.2.1.3 The Feature Model

A developed and documented Commonality and Variability Analysis (CVA) during the requirements collection phase may give developers an insight into what roles might be appropriate for the multi-agent system to be developed. In terms of multi-agent system (MAS) development, a CVA may assist in the identification of possible roles since it partitions those requirements that will be found in every future instantiation of a particular role from those requirements that will only be found in some instantiations of a particular role.

The actual identification of appropriate roles for a MAS is not discussed here. Gaia proposes to identify roles through an inspection of the problem (via the division of a system into organizations and sub-organizations) [92], [94]. Rather, for Gaia-PL we only claim that documenting a MAS requirements in a CVA may aid in confirming the role definition and help in the preliminary role model(s).

In the collection of the requirements for the Prospecting Asteroid Mission (PAM) case study used in this dissertation, we found that it was straightforward to group both the commonality and variability requirements into logical, functional groups. As detailed in Chapter 3, the PAM mission relies on four autonomous characteristics to operate: self-coordination, self-healing, self-optimization and self-protection. Thus, it was natural to identify and group requirements in such categories for both commonality requirements and variability requirements. In addition, it was useful to group variable requirements into groups depending on what type of spacecraft the requirements were targeted for (i.e., a *Leader*, *Messenger* or *Worker* spacecraft of the PAM swarm). Such groupings of the requirements in the CVA for a MAS may also provide guidance to the identification of the roles for the agents of a MAS, as was the case in our PAM case study.

The variabilities of the CVA will help define the variation points of the product-line, multi-agent system. Partitioning the variabilities into similar groups provides the

initial requirements for the variation points of a system. For example, from Figure 6 we can derive the variation points for the *Self-Optimizer* role, discussed in Section 4.1.2.1. The variability V_SO2 implies the “leader” variation point, variabilities V_SO3 and V_SO4 imply the “messenger” variation point and variability V_SO6 implies the “worker” variation point all for the *Self-Optimizer* role. Similarly, from Figure 5 we can derive the common functionality for the *Self-Optimizer* role from commonalities C_SO1, C_SO2, C_SO3 and C_SO4.

In addition to a CVA, this work utilized a Feature Model, shown in Figure 9 as well as in Appendix C, to help identify and organize the roles and variation points of the PAM case study. Using the CVA, requirements can be further refined and detailed requirements can be derived during the analysis and design phases so that a Feature Model and more detailed requirements specifications can be created and documented [43], [67], [80]. Pohl, Böckle and van der Linden have provided a process to derive a Feature Model from the requirements of a CVA [67]. Thus, we do not cover this process here.

However, from the Feature Model, shown in Figure 9 and in Appendix C, the roles and variation points are readily illustrated. For example, the *Self-Optimizer* role, discussed in Section 4.1.2.1, is shown as a mandatory feature of a PAM spacecraft in which only one of the subfeatures (i.e., variation points) “optimization for workers”, “optimization for messengers” or “optimization for leader” may be selected. For the *LeaderPlanner* role, also discussed in Section 4.1.2.1, the Feature Model illustrates this as the Leader and Planning subfeatures a subfeature of the Swarm Role feature. As indicated in the Feature Model, of the variation points for this role, “passive” and “active” at least one must be selected. This follows exactly how the *LeaderPlanner* role was described in Section 4.1.2.1.

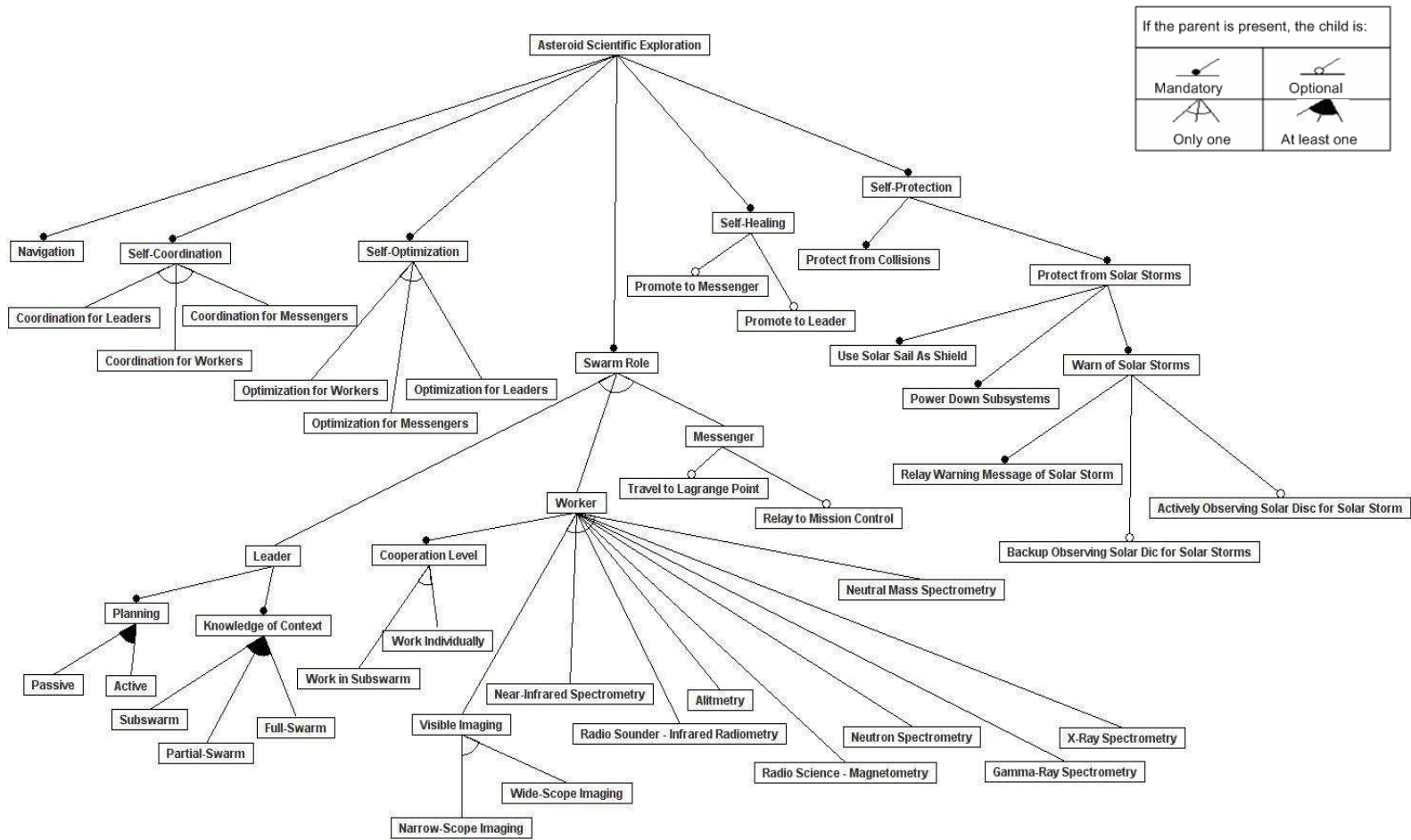


Figure 9 Feature Model Derived from the Commonality and Variability Analysis for the PAM MAS-PL

4.2.2 Analysis and Design Phase

The Analysis and Design Phase of Gaia-PL takes the requirements documented in the Requirements Documentation Phase and develops and documents the multi-agent system product line's (MAS-PL) requirements specifications. Requirements specifications are documented in three schemas: The Role Schema, The Role Variation Points Schema and The Variation Point Schema. These schemas serve as a requirements specification pattern in which requirements can be defined and documented.

This section describes the development and documentation of the roles and variation points for the Prospecting Asteroid Mission (PAM) from the requirements discussed in the previous section and documented in the Commonality and Variability Analysis in Appendix A. Note that the complete set of schemas documenting PAM mission's requirements specifications can be found in Appendix D. In this section, we only show a small set of the schemas to illustrate Gaia-PL.

4.2.2.1 The Role Schema

For those roles that have been identified having no variation points (i.e., the role will have identical functionality in all agents that have the role), Gaia-PL uses a slightly modified version of the Role Schema from Gaia [92], [94]. For example, the *Navigator* role of the PAM mission, discussed in Section 4.1.2.1, was identified to have no variation points and thus can be documented in Gaia's Role Schema, shown in Figure 10.

The process Role Schema used to document the requirements specifications in Gaia-PL for those roles that have no variation points is identical to Gaia [92], [94] and is therefore not discussed here. However, Gaia-PL does include additional information into the requirements specifications schemas. First, we introduce identification numbers to all schemas for traceability, organization and management purposes. Second, a row is added to indicate specifically which variation point the requirements specification is describing

(not applicable for a Role Schema, however, since there are no associated variation points). An “Inherits” row provides which schemas must be included with the schema for a particular variation point. This will be described further in the following section. Finally, rows to indicate the Parameters of Variation and Requirements that are related to the schema are provided also for traceability, organization and management purposes.

Role Schema: Navigator	Schema ID: N
Variation Point: N/A	
Inherits: None	
Parameters of Variation: N/A	
Requirements: C_M1, C_M2, C_M3, C_M4, C_M5, C_M6, C_M7, C_M8	
Description: Provides the functionality to a spacecraft to maneuver itself using its solar sail.	
Activities and Protocols: AdjustSolarSail, CalculateThrust, CheckOrbit, CheckSolarSailStatus, CheckSystemStatus, ExtendSolarSail, MoveToPosition, RetractSolarSail	
Permissions:	
Reads -	
<i>currentAttitude</i>	// attitude of the spacecraft
<i>currentOrbit</i>	// current orbit of the spacecraft
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>systemStatus</i>	// status of the spacecraft
<i>solarSailStatus</i>	// status of the solar sail
Changes -	
<i>currentAttitude</i>	// attitude of the spacecraft
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
Generates -	
<i>systemStatus</i>	// status of the spacecraft
<i>solarSailStatus</i>	// status of the solar sail
<i>thrustNeeded</i>	// calculated thrust needed to move
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to maneuver the spacecraft to the desired location.	
Safety -	
None.	

Figure 10 The Requirements Specifications for the Navigator Role Documented in a Role Schema

Role Variation Points Schema: SelfOptimizer		Schemata ID: SO
Parameters of Variation: P4, P5, P6		
Description:		
At the swarm-level, the collection of these roles within all the spacecraft aids in autonomously and continuously improving the spacecraft's ability to identify, explore and communicate the information discovered while investigating asteroids. At the spacecraft-level, these roles aid in the spacecraft to continuously learn and improve its specialized abilities and communicate its findings with other similar spacecraft.		
Variation Points:		
<u>Core:</u>	The core elements of a spacecraft to be able to optimize itself in regard to general spacecraft functions so that it can continuously learn from the environment and perform better within the swarm. [SO-Core]	
Leader:	The elements needed in a <i>leader</i> spacecraft to be able to optimize itself in regards to its ability to best manage, oversee and direct the swarm to optimize the swarm's ability to achieve scientific goals. [SO-Leader]	
Messenger:	The elements needed in a <i>messenger</i> spacecraft to be able to optimize itself in regards to its ability to best perform the communication necessary within the swarm so that commands and information can best be transmitted. [SO-Messenger]	
Worker:	The elements needed in a <i>worker</i> spacecraft to be able to optimize itself in regards to its ability to best optimize its ability to achieve its own scientific goals. [SO-Worker]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time. However, a spacecraft that may switch it's operating variation point (i.e., P2=True or P3=True) may have this variation point alter at runtime.		

Figure 11 The Role Variation Points Schema for the Self-Optimizer Role

4.2.2.2 The Role Variation Points Schema

The Role Variation Schema, shown in Figure 11 for the *Self-Optimizer* role discussed in Section 4.1.2.1, defines a role and the variation points that the role can assume during its lifetime. The Role Variation Point Schema, introduced in Gaia-PL, describes the role, the role's variation points and the binding time for the variation points. The variation points are described for the role and provide the identification tags (e.g., SO-Core, SO-Leader, etc.) for the Variation Point Schema, discussed in the next section, to aid in traceability, organization and management of the requirements, parameters of variation, roles and variation points of the multi-agent system product line (MAS-PL).

For most roles, one of the variation points listed in the Role Variation Point Schema will contain the common functionality of the role, denoted in the Role Variation Point Schema by being underlined. Thus, this variation point will be included for all agents containing the role in addition to the other selected variation point(s). For example, the “Core” variation point for the *Self-Optimizer* role shown in Figure 11 contains the common functionality (i.e., commonality requirements C_SO1, C_SO2, C_SO3 and C_SO4 from the Commonality and Variability Analysis (CVA)) for the role.

The introduction of the Role Variation Point Schema in Gaia-PL provides software engineers with the ability to define a role in a hierarchical manner. The common functionality defined by a variation point (e.g., the “Core” variation point for the *Self-Optimizer* role shown in Figure 11) is further refined by the variable variation points. Thus, the Role Variation Point Schema achieves the hierarchical nature of the functionality in a role as modeled by the Feature Model, see Figure 9.

4.2.2.3 The Variation Point Schema

The Variation Point Schema, shown in Figure 12, Figure 13, Figure 14 and Figure 15 for the variation points of the *Self-Optimizer* role, captures the requirements of a role variation point's capabilities. The Variation Point Schema and the Role Schema, described in Section 4.2.2.1 are identical; however, the Variation Point Schema will always have a Role Variation Points Schema associated with it (denoted in the Schema-ID using the convention of *Role Variation Points Schema ID – Variation Point ID*). Some variation points will inherit other variation points, as denoted in the Inherits row. For example, the Variation Point Schema in Figure 13 denotes that it inherits the SO-Core variation point, Figure 12, since the SO-Core Variation Point Schema provides the common functionality of the *Self-Optimizer* role. This additionally illustrates the hierarchical nature possible in the definition of a role in the Gaia-PL methodology.

Role Schema: SelfOptimizer	Schema ID: SO-Core																																
Variation Point: Core																																	
Inherits: None																																	
Parameters of Variation: N/A																																	
Requirements: C_SO1, C_SO2, C_SO3, C_SO4, C_M1, C_M2, C_M4, C_M5																																	
Description: Provides the spacecraft with the functionality to optimize itself in regards to general spacecraft functions so that it can continuously learn from the environment and perform better within the swarm.																																	
Activities and Protocols: AdjustToEnviron, CalcNewPosition, CalibrateInstr, CheckSystemStatus, CheckEnvironStatus, CheckPowerConsump, CheckSolarCellStatus, EvaluatePositionForGoal, MoveNewPos																																	
Permissions: Reads - <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;"><i>currentAttitude</i></td> <td style="padding-left: 20px;">// current attitude of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentGoal</i></td> <td style="padding-left: 20px;">// current goal of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentPosition</i></td> <td style="padding-left: 20px;">// current position of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentVelocityIncr</i></td> <td style="padding-left: 20px;">// current velocity increment of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>environmentStatus</i></td> <td style="padding-left: 20px;">// current status of the detectable parts of the surrounding environment</td> </tr> <tr> <td style="padding-left: 20px;"><i>powerConsumpLevel</i></td> <td style="padding-left: 20px;">// current level of the spacecraft's power consumption</td> </tr> <tr> <td style="padding-left: 20px;"><i>riskForSystemFactor</i></td> <td style="padding-left: 20px;">// current risk to spacecraft to see if recent solar storm</td> </tr> <tr> <td style="padding-left: 20px;"><i>systemStatus</i></td> <td style="padding-left: 20px;">// current status of the spacecraft</td> </tr> </table> Changes - <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;"><i>environmentState</i></td> <td style="padding-left: 20px;">// current state that the spacecraft believes its surrounding environment is in</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentPosition</i></td> <td style="padding-left: 20px;">// current position of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentAttitude</i></td> <td style="padding-left: 20px;">// current attitude of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentVelocityIncr</i></td> <td style="padding-left: 20px;">// current velocity increment of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>instrCalibValue</i></td> <td style="padding-left: 20px;">// vector of the current calibration values for the onboard instruments</td> </tr> <tr> <td style="padding-left: 20px;"><i>instrVector</i></td> <td style="padding-left: 20px;">// vector of all the spacecraft's onboard instruments</td> </tr> </table> Generates - <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;"><i>newEnvironStatus</i></td> <td style="padding-left: 20px;">// new status of the detectable parts of the surrounding environment</td> </tr> <tr> <td style="padding-left: 20px;"><i>newVelocityIncr</i></td> <td style="padding-left: 20px;">// calculated new velocity increment for the spacecraft</td> </tr> </table>		<i>currentAttitude</i>	// current attitude of the spacecraft	<i>currentGoal</i>	// current goal of the spacecraft	<i>currentPosition</i>	// current position of the spacecraft	<i>currentVelocityIncr</i>	// current velocity increment of the spacecraft	<i>environmentStatus</i>	// current status of the detectable parts of the surrounding environment	<i>powerConsumpLevel</i>	// current level of the spacecraft's power consumption	<i>riskForSystemFactor</i>	// current risk to spacecraft to see if recent solar storm	<i>systemStatus</i>	// current status of the spacecraft	<i>environmentState</i>	// current state that the spacecraft believes its surrounding environment is in	<i>currentPosition</i>	// current position of the spacecraft	<i>currentAttitude</i>	// current attitude of the spacecraft	<i>currentVelocityIncr</i>	// current velocity increment of the spacecraft	<i>instrCalibValue</i>	// vector of the current calibration values for the onboard instruments	<i>instrVector</i>	// vector of all the spacecraft's onboard instruments	<i>newEnvironStatus</i>	// new status of the detectable parts of the surrounding environment	<i>newVelocityIncr</i>	// calculated new velocity increment for the spacecraft
<i>currentAttitude</i>	// current attitude of the spacecraft																																
<i>currentGoal</i>	// current goal of the spacecraft																																
<i>currentPosition</i>	// current position of the spacecraft																																
<i>currentVelocityIncr</i>	// current velocity increment of the spacecraft																																
<i>environmentStatus</i>	// current status of the detectable parts of the surrounding environment																																
<i>powerConsumpLevel</i>	// current level of the spacecraft's power consumption																																
<i>riskForSystemFactor</i>	// current risk to spacecraft to see if recent solar storm																																
<i>systemStatus</i>	// current status of the spacecraft																																
<i>environmentState</i>	// current state that the spacecraft believes its surrounding environment is in																																
<i>currentPosition</i>	// current position of the spacecraft																																
<i>currentAttitude</i>	// current attitude of the spacecraft																																
<i>currentVelocityIncr</i>	// current velocity increment of the spacecraft																																
<i>instrCalibValue</i>	// vector of the current calibration values for the onboard instruments																																
<i>instrVector</i>	// vector of all the spacecraft's onboard instruments																																
<i>newEnvironStatus</i>	// new status of the detectable parts of the surrounding environment																																
<i>newVelocityIncr</i>	// calculated new velocity increment for the spacecraft																																
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the spacecraft's ability to perform its given tasks. Safety - None.																																	

Figure 12 The Variation Points Schema for the Core Variation Point of the Self-Optimizer Role

Role Schema: SelfOptimizer	Schema ID: SO-Leader																		
Variation Point: Leader																			
Inherits: SO-Core																			
Requirements: V_SO1, V_SO2, C_SC1, C_SC2, V_L6, V_L7, V_L8, V_L11																			
Parameters of Variation: P4=True																			
<p>Description: Provides the spacecraft with the elements needed in a <i>leader</i> spacecraft to be able to optimize itself in regard to its ability to manage, oversee and direct the swarm to optimize the swarm's ability to achieve scientific goals. Specifically, the ability for a <i>leader</i> spacecraft to optimize its ability to identify asteroids of interest and share this information.</p>																			
<p>Activities and Protocols: DeviseNewAsteroidIdRules, EvaluateCurrentAsteroidIdRules, ReviewAsteroidIdHis, <u>AcceptOptimizationInfo</u>, <u>AcceptOptimizationReq</u>, <u>RequestOptimizationInfo</u>, <u>ShareOptimizationInfo</u></p>																			
<p>Permissions:</p> <p>Reads -</p> <table> <tr> <td><i>asteroidIdRules</i></td> <td>// current vector of rules that is used to // identify asteroids of interest given // preliminary data points on the asteroid</td> </tr> <tr> <td><i>asteroidPrelimData</i></td> <td>// preliminary data points of an asteroid</td> </tr> <tr> <td><i>asteroidId</i></td> <td>// identification number of an asteroid</td> </tr> <tr> <td><i>asteroidIdHistory</i></td> <td>// the history log kept of the spacecraft's // identification of asteroids of interest</td> </tr> <tr> <td><i>optimizationInfoRec</i></td> <td>// message to received after requesting // for another spacecraft's current // optimization information</td> </tr> <tr> <td><i>leaderVector</i></td> <td>// vector of nearby <i>leader</i> spacecraft // to aid in sharing optimization information</td> </tr> </table> <p>Changes -</p> <table> <tr> <td><i>asteroidIdRules</i></td> <td>// vector of rules that is used to identify // asteroids of interest given preliminary // data points on the asteroid</td> </tr> </table> <p>Generates -</p> <table> <tr> <td><i>asteroidIdRulesValue</i></td> <td>// evaluation value of the accuracy of the // spacecraft's current ability to identify // asteroids of interest</td> </tr> <tr> <td><i>optimizationInfoMsg</i></td> <td>// message to deliver upon receiving a // request for spacecraft's current // optimization information</td> </tr> </table>		<i>asteroidIdRules</i>	// current vector of rules that is used to // identify asteroids of interest given // preliminary data points on the asteroid	<i>asteroidPrelimData</i>	// preliminary data points of an asteroid	<i>asteroidId</i>	// identification number of an asteroid	<i>asteroidIdHistory</i>	// the history log kept of the spacecraft's // identification of asteroids of interest	<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information	<i>leaderVector</i>	// vector of nearby <i>leader</i> spacecraft // to aid in sharing optimization information	<i>asteroidIdRules</i>	// vector of rules that is used to identify // asteroids of interest given preliminary // data points on the asteroid	<i>asteroidIdRulesValue</i>	// evaluation value of the accuracy of the // spacecraft's current ability to identify // asteroids of interest	<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information
<i>asteroidIdRules</i>	// current vector of rules that is used to // identify asteroids of interest given // preliminary data points on the asteroid																		
<i>asteroidPrelimData</i>	// preliminary data points of an asteroid																		
<i>asteroidId</i>	// identification number of an asteroid																		
<i>asteroidIdHistory</i>	// the history log kept of the spacecraft's // identification of asteroids of interest																		
<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information																		
<i>leaderVector</i>	// vector of nearby <i>leader</i> spacecraft // to aid in sharing optimization information																		
<i>asteroidIdRules</i>	// vector of rules that is used to identify // asteroids of interest given preliminary // data points on the asteroid																		
<i>asteroidIdRulesValue</i>	// evaluation value of the accuracy of the // spacecraft's current ability to identify // asteroids of interest																		
<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information																		
<p>Responsibilities:</p> <p>Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to identify asteroids of interests to investigate for all <i>leader</i> spacecraft in the swarm.</p> <p>Safety - None.</p>																			

Figure 13 The Variation Points Schema for the Leader Variation Point of the Self-Optimizer Role

Role Schema: SelfOptimizer	Schema ID: SO-Messenger														
Variation Point: Messenger															
Inherits: SO-Core															
Requirements: V_SO3, V_SO4, C_SC1, C_SC2															
Parameters of Variation: P5=True															
Description: Provides the spacecraft with the elements needed in a <i>messenger</i> spacecraft to be able to optimize itself in regards to its ability to perform the communication necessary within the swarm so that commands and information can best be transmitted. Specifically, the ability of the spacecraft to optimize its positioning for communications and sharing this information with others.															
Activities and Protocols: DeviseNewCommStrategy, EvaluateCurrentCommStrategy, EvaluateCurPosition, ReviewCommHis, <u>AcceptOptimizationInfo</u> , <u>AcceptOptimizationReq</u> , <u>RequestOptimizationInfo</u> , <u>ShareOptimizationInfo</u>															
Permissions: Reads - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>communicationStrategy</i></td> <td style="padding-left: 20px;">// current strategy for spacecraft's // communication</td> </tr> <tr> <td style="padding-left: 20px;"><i>communicationHist</i></td> <td style="padding-left: 20px;">// current history log of the spacecraft's // past communication sessions</td> </tr> <tr> <td style="padding-left: 20px;"><i>optimizationInfoRec</i></td> <td style="padding-left: 20px;">// message to received after requesting // for another spacecraft's current // optimization information</td> </tr> <tr> <td style="padding-left: 20px;"><i>messengerVector</i></td> <td style="padding-left: 20px;">// vector of nearby <i>messenger</i> spacecraft // to aid in sharing optimization information</td> </tr> </table> Changes - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>communicationStrategy</i></td> <td style="padding-left: 20px;">// current strategy for spacecraft's // communication</td> </tr> </table> Generates - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>optimizationInfoMsg</i></td> <td style="padding-left: 20px;">// message to deliver upon receiving a // request for spacecraft's current // optimization information</td> </tr> <tr> <td style="padding-left: 20px;"><i>communicationStratVal</i></td> <td style="padding-left: 20px;">// evaluation value of the accuracy of the // spacecraft's current ability to // communicate with the subswarm</td> </tr> </table>		<i>communicationStrategy</i>	// current strategy for spacecraft's // communication	<i>communicationHist</i>	// current history log of the spacecraft's // past communication sessions	<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information	<i>messengerVector</i>	// vector of nearby <i>messenger</i> spacecraft // to aid in sharing optimization information	<i>communicationStrategy</i>	// current strategy for spacecraft's // communication	<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information	<i>communicationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // communicate with the subswarm
<i>communicationStrategy</i>	// current strategy for spacecraft's // communication														
<i>communicationHist</i>	// current history log of the spacecraft's // past communication sessions														
<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information														
<i>messengerVector</i>	// vector of nearby <i>messenger</i> spacecraft // to aid in sharing optimization information														
<i>communicationStrategy</i>	// current strategy for spacecraft's // communication														
<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information														
<i>communicationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // communicate with the subswarm														
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to communicate for all <i>messenger</i> spacecraft in the swarm. Safety - None.															

Figure 14 The Variation Points Schema for the Messenger Variation Point of the Self-Optimizer Role

Role Schema: SelfOptimizer	Schema ID: SO-Worker
Variation Point: Worker	
Inherits: SO-Core	
Requirements: V_SO5, V_SO6, C_SC1, C_SC2	
Parameters of Variation: P6=True	
Description: The elements needed in a <i>worker</i> spacecraft to be able to optimize itself in regards to its ability to best optimize its ability to achieve its own scientific goals.	
Activities and Protocols: DeviseNewSciExplorStrategy, EvaluateCurrentSciExplorStrategy, EvaluateCurPosition, ReviewSciExplorHis, <u>AcceptOptimizationInfo</u> , <u>AcceptOptimizationReq</u> , <u>RequestOptimizationInfo</u> , <u>ShareOptimizationInfo</u>	
Permissions:	
Reads -	
<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information
<i>sciExplorationStrategy</i>	// current strategy for spacecraft's // science exploration using its specialized // onboard equipment
<i>sciExplorationRules</i>	// current rules for the spacecraft to abide // by in its scientific exploration
<i>sciExplorationHist</i>	// current history log of the spacecraft's // past science exploration of asteroids
<i>workerType</i>	// the type of worker spacecraft (i.e., based // on its specialized onboard equipment
<i>workerVector</i>	// vector of nearby <i>worker</i> spacecraft with // the same onboard equipment
<i>scienceGoal</i>	// current scientific goal pursued by the // spacecraft
Changes -	
<i>sciExplorationStrategy</i>	// strategy for spacecraft's science // exploration using its specialized onboard // equipment
Generates -	
<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information
<i>sciExplorationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // achieve its scientific goals
Responsibilities:	
Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to achieve scientific goals for all similar <i>worker</i> spacecraft in the swarm.	
Safety - None.	

Figure 15 The Variation Points Schema for the Worker Variation Point of the Self-Optimizer Role

4.2.2.3 Documenting the Roles and Variation Points in Gaia-PL

During the initial development of a multi-agent system product line (MAS-PL) (the product-line domain engineering phase of the Family-Oriented Abstraction, Specification and Translation (FAST) product-line methodology [88]), the focus must be primarily on identifying the overall requirements specifications of the system. It is later (during the product-line application engineering phase of FAST) that actual members of the distributed system can be instantiated with some or all of the requirements established earlier. We consider those initial requirement specifications in the Role Variation Points Schema and the Variation Point Schema. Note that, the Role Schema is not discussed here since its documentation follows that of Gaia.

To capture the requirements specifications of the roles and variation points of a MAS-PL and document them in the two schemas, we use the following procedure:

1. Identify the roles within the system as discussed in Section 4.2.1. Each role will constitute a new Role Variation Points Schema to be created. If the role has no identified variation points (see Step 3), then simply create a new Role Schema and follow Steps 4a – 4c).
2. For each role, provide the role's name, a unique identification, a listing of the associated parameters of variation, a brief description of the role and the variation points binding time in the appropriate fields of the Role Variation Points Schema. In Gaia-PL we follow and advocate the naming and numbering scheme of Schetter, Campbell and Surka from [73] as shown for the for the *Self-Optimizer* role depicted in Figure 11.
3. For each role, identify and define the differing variation points that the role can adopt during all envisioned execution scenarios of the system as described in Section 4.2.1. For each variation point, fill in the Variation

Points section of the Role Variation Points Schema by including the name, a brief description of the variation point and a reference identification number to the Role Variation Point Schema that gives the detailed requirements of the variation point (see Step 4a).

4. For each identified variation point (Step 3), create a new Variation Point Schema. For each Variation Point Schema:
 - a. Document the name of the role to which the variation point corresponds as well as the name of the variation points in the appropriate sections of the Variation Point Schema. Indicate the variation point identification tag (corresponding to the variation point identification in Step 3) in the appropriate field in the Role Variation Points Schema. Further, provide the identification tags of the associated product-line requirements and parameters of variation as well as an identification tag to any Variation Point Schema(s) or Role Schema that the variation point inherits.
 - b. Identify the protocols, activities, permissions and responsibilities that are particular to only that variation point. That is, define the protocols, activities, permissions and responsibilities that are not found in any of the variation points.
 - c. Document and define the identified protocols, activities, permissions and responsibilities in the appropriate sections of the Role Variation Point. (Note, in accordance with the Gaia conventions, activities are distinguished from protocols by being underlined in Gaia-PL).

These steps result in a set of Role Variation Points Schemas that have an associated set of Role Variation Point Schemas. Additionally, these steps conform to the domain engineering phase of software product-line development in that they define the

MAS-PL's requirements, design, architecture and other software engineering assets that pertain to all of the agents, rather than to just a single type of agent [88].

Figure 11, Figure 12, Figure 13, Figure 14 and Figure 15 illustrated the Role Variation Point Schema and the Variation Points Schema for the *Self-Optimizer* role of the PAM case study used in this dissertation. The entire PAM case study developed and documented a total of 11 Role Variation Points Schemas and 39 Variation Points Schemas that can be found in Appendix A. To further illustrate the Gaia-PL approach to designing and documenting the requirements specifications of a MAS-PL in the Role Variation Points Schema and the Variation Points Schema, we provide additional examples to illustrate some minor differences in the roles and variation points discovered in the PAM case study. Figure 16 shows a portion of the full Feature Model of the PAM case study from Figure 9 that describes the *Self-Protector* role. (Note that the *Self-Protector* role additionally includes functionality to prevent collisions that is not

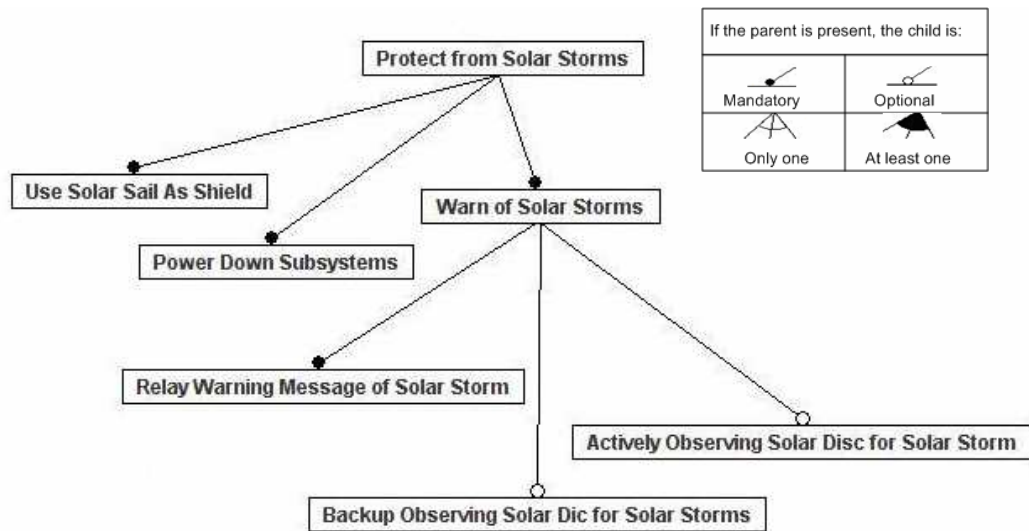


Figure 16 A Portion of the PAM Feature Model to Illustrate Hierarchical Role Variation Points Schemas

discussed here or shown in Figure 16). Further analysis of the requirements for this role and feature revealed two additional subroles of the *Self-Protector* role were required: a *SolarStormWarner* role and a *SolarStormProtector* role. Additionally, the *SolarStormWarner* role had three associated variation points.

To document the requirements specifications for these roles while maintaining the structure of the Feature Model, the Role Variation Points Schema for the *Self-Protector* role was defined with variation points for the *SolarStormWarner* and *SolarStormProtector* roles, shown in Figure 17. Note that these are required roles (i.e., variation points) according to the Feature Model. The identification tag given for the roles, SSW and SSP respectively, identify the Role Variation Points Schema and Variation Point Schema for the *SolarStormWarner* and *SolarStormProtector* roles. The *SolarStormWarner* Role Variation Points Schema, shown in Figure 18, then lists the possible variation points (Figure 19, Figure 20 and Figure 21) for the role similar to the *Self-Optimizer* role example shown in Figure 11 and described above. The *SolarStormProtector* role (Figure 22), however, does not contain any variation points (i.e., the functionality listed will be identical for all agents with the *SolarStormProtector* role) and, thus, defines the role's functionality only in a Role Schema similar to the *Navigator* role example shown in Figure 10 and described above.

This situation encountered in the PAM case study illustrates the need for the ability of a software engineer to define a role's requirements specifications hierarchically, a feature of Gaia-PL not possible in Gaia. Here, the ability to define the *Self-Protector* role hierarchically allows the requirements specifications to more accurately reflect the MAS-PL's Feature Model and avoid potential confusion amongst the relationship(s) of the roles, variation points and requirements of a complex system.

Role Variation Points Schema: SelfProtector		Schemata ID: SP
Parameters of Variation: N/A		
Description:		
At the swarm-level, the collection of these roles within all the spacecraft aid in autonomously maintaining the system's scientific operations while enduring solar storms, spacecraft collisions, etc.		
Variation Points:		
SolarStormWarner:	Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm. [SSW]	
SolarStormProtector:	Protects the spacecraft from the solar radiation present during solar storms by using the solar sail as a shield, powering off systems and/or moving to a better position. [SSP]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time,		

**Figure 17 An Excerpt of the Role Variation Points Schema
for the Self-Protector Role**

Role Variation Points Schema: SolarStormWarner		Schemata ID: SSW
Parameters of Variation: P7, P8		
Description:		
Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm.		
Variation Points:		
<u>Passive:</u>	The spacecraft does not have the ability to constantly monitor the solar disc to watch for solar storms but can warn other spacecraft after itself receiving a warning message. [SSW-Passive]	
Warm-Spare:	The spacecraft has the ability to constantly monitor the solar disc to watch for solar storms and receive messages from mission control but is acting in a backup/redundant capacity. [SSW-Warm]	
Active:	The spacecraft is tasked to constantly monitor the solar disc and receive warning messages from mission control so that it can warn other spacecraft of an impending solar storm. [SSW-Active]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time, however, the spacecraft may switch its operating variation point (e.g., from Warm-Spare to Active) at runtime. All spacecraft shall have the Passive variation point as a commonality. Spacecraft with the Warm-Spare variation point shall also include all functionality of Passive.		

**Figure 18 An Excerpt of the Role Variation Points Schema for the
SolarStormWarner Role**

Role Schema: SolarStormWarner	Schema ID: SSW-Passive
Variation Point: Passive	
Inherits: SP-Core	
Parameters of Variation: P7=Passive; P8=False	
Requirements: C_G1, C_SH4, C_SP5, C_SP8, V_SP1, V_SP2	
Description: Receives warnings from other spacecraft about impending solar storms and calculates the risk factor to itself from solar radiation damage. Notifies other nearby spacecraft of the impending solar storm.	
Activities and Protocols: CalculateStormRisk, UpgradeToWarm, <u>AcceptUpgrade</u> , <u>AcceptWarnMsg</u> , <u>RecieveHeartbeat</u> , <u>ReplyHeartBeat</u> , <u>SendSolarStormWarnMsg</u>	
Permissions: Reads - <ul style="list-style-type: none"> <i>position</i> // current spacecraft position <i>velocityIncrement</i> // current spacecraft velocity increment <i>curScienceGoalFactor</i> // current spacecraft scientific goal factor <i>subswarmVector</i> // vector of nearby spacecraft to warn supplied <i>stormType</i> // type of storm supplied by warning supplied <i>stormIntensity</i> // storm intensity supplied by warning supplied <i>stormVector</i> // storm vector supplied by warning Changes - <ul style="list-style-type: none"> <i>riskForSystemFactor</i> // current risk to spacecraft Generates - <ul style="list-style-type: none"> <i>stormRiskValue</i> // new value of the risk to the spacecraft of // the solar storm 	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to satisfy scientific goals while minimizing the risk factor. Safety - Prevent other spacecraft from being damaged by notifying others.	

Figure 19 The Variation Points Schema for the Passive Variation Point of the SolarStormWarner Role

Role Schema: SolarStormWarner	Schema ID: SSW-Active																						
Variation Point: Active																							
Inherits: SSW-Warm																							
Parameters of Variation: P7=Active; P8=True																							
Requirements: C_M9, V_SP1, V_SP2																							
<p>Description:</p> <p>Continuously monitors the solar disc for the signs of an impending solar storm whose solar radiation may damage the swarm's spacecraft. Upon detecting a solar storm, it seeks to verify the data and then proceeds to warn the swarm's spacecraft. Also able to receive warning messages from mission control of an impending solar storm.</p>																							
<p>Activities and Protocols:</p> <p>CompareMissionControlData, DowngradeToWarm, AcceptDowngrade, AcceptMissionControlWarn, <u>AcceptStormDataVote</u>, <u>InitiateStormDataVote</u>, <u>InitiateStromWarning</u></p>																							
<p>Permissions:</p> <p>Reads -</p> <table> <tr> <td><i>detectedStormType</i></td> <td>// type of storm as detected</td> </tr> <tr> <td><i>detectedStormIntensity</i></td> <td>// intensity of the storm as detected</td> </tr> <tr> <td><i>detectedStormVector</i></td> <td>// storm vector as detected</td> </tr> <tr> <td>supplied <i>MCStormType</i></td> <td>// type of storm supplied by mission control</td> </tr> <tr> <td>supplied <i>MCStormIntensity</i></td> <td>// storm intensity supplied by mission control</td> </tr> <tr> <td>supplied <i>MCstormVector</i></td> <td>// storm vector supplied by mission control</td> </tr> </table> <p>Changes -</p> <table> <tr> <td><i>stormRiskValue</i></td> <td>// new value of the risk to the spacecraft of the solar storm</td> </tr> </table> <p>Generates -</p> <table> <tr> <td><i>riskForSystemFactor</i></td> <td>// current risk to spacecraft</td> </tr> <tr> <td><i>stromWarningConfidence</i></td> <td>// confidence in the warning provided by mission control</td> </tr> <tr> <td><i>voteConfidence</i></td> <td>// confidence in the verification of detected storm data by other spacecraft</td> </tr> <tr> <td><i>warningMessage</i></td> <td>// warning message to be sent to other spacecraft</td> </tr> </table>		<i>detectedStormType</i>	// type of storm as detected	<i>detectedStormIntensity</i>	// intensity of the storm as detected	<i>detectedStormVector</i>	// storm vector as detected	supplied <i>MCStormType</i>	// type of storm supplied by mission control	supplied <i>MCStormIntensity</i>	// storm intensity supplied by mission control	supplied <i>MCstormVector</i>	// storm vector supplied by mission control	<i>stormRiskValue</i>	// new value of the risk to the spacecraft of the solar storm	<i>riskForSystemFactor</i>	// current risk to spacecraft	<i>stromWarningConfidence</i>	// confidence in the warning provided by mission control	<i>voteConfidence</i>	// confidence in the verification of detected storm data by other spacecraft	<i>warningMessage</i>	// warning message to be sent to other spacecraft
<i>detectedStormType</i>	// type of storm as detected																						
<i>detectedStormIntensity</i>	// intensity of the storm as detected																						
<i>detectedStormVector</i>	// storm vector as detected																						
supplied <i>MCStormType</i>	// type of storm supplied by mission control																						
supplied <i>MCStormIntensity</i>	// storm intensity supplied by mission control																						
supplied <i>MCstormVector</i>	// storm vector supplied by mission control																						
<i>stormRiskValue</i>	// new value of the risk to the spacecraft of the solar storm																						
<i>riskForSystemFactor</i>	// current risk to spacecraft																						
<i>stromWarningConfidence</i>	// confidence in the warning provided by mission control																						
<i>voteConfidence</i>	// confidence in the verification of detected storm data by other spacecraft																						
<i>warningMessage</i>	// warning message to be sent to other spacecraft																						
<p>Responsibilities:</p> <p>Liveness -</p> <p>If the spacecraft is functioning properly, this role will eventually be able to establish a communication connection with mission control.</p> <p>Safety -</p> <p>Initiate warnings to spacecraft of an impending solar storm.</p>																							

Figure 21 The Variation Points Schema for the Active Variation Point of the SolarStormWarner Role

Role Schema: SolarStormProtector	Schema ID: SSP
Variation Point: SolarStormProtector	
Inherits: None	
Parameters of Variation: N/A	
Requirements: C_SP5, C_SP6, C_SP7	
Description: Provides the spacecraft with the functionality to autonomously protect itself from the affects of solar radiation during a solar storm.	
Activities and Protocols: CheckSolarSailStatus, DeploySolarSailAsShield, EvaluateRiskToGoal, PowerDownSubsystems, PowerUpSubsystems	
Permissions:	
Reads -	
<i>curScienceGoalFactor</i>	// current spacecraft scientific goal factor
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
<i>solarSailStatus</i>	// status of the solar sail
<i>detectedStormType</i>	// type of storm as detected
<i>detectedStormIntensity</i>	// intensity of the storm as detected
<i>detectedStormVector</i>	// storm vector as detected
<i>subsystemsList</i>	// vector list of the spacecraft's subsystems
Changes -	
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
<i>systemStatus</i>	// status of the spacecraft
<i>solarSailStatus</i>	// status of the solar sail
<i>subsystemsStatus</i>	// list of the statuses of the spacecraft's subsystems
Generates -	
<i>riskForSystemFactor</i>	// current risk to spacecraft
<i>riskToGoalFactor</i>	// calculated value of the current risk factor to the advantage of pursuing scientific exploration
Responsibilities:	
Liveness - If the spacecraft is functioning properly, this role will eventually take the steps needed to prevent radiation damage from a solar storm.	
Safety - Prevent the solar radiation damage to the spacecraft possible during a solar storm.	

Figure 22 The Variation Points Schema for the SolarStormProtector Variation Point of the Self-Protector Role

The Analysis and Design Phase of Gaia-PL takes the requirements documented in the Requirements Documentation Phase and develops and documents the multi-agent system product line's (MAS-PL) requirements specifications. Requirements specifications are documented in three schemas: The Role Schema, The Role Variation Points Schema and The Variation Point Schema. These schemas serve as a requirements specification pattern in which requirements can be defined and documented.

This section describes the development and documentation of the roles and variation points for the Prospecting Asteroid Mission (PAM) from the requirements discussed in the previous section and documented in the Commonality and Variability Analysis in Appendix A. Note that the complete set of schemas documenting PAM mission's requirements specifications can be found in Appendix D. In this section, we only show a small set of the schemas to illustrate Gaia-PL.

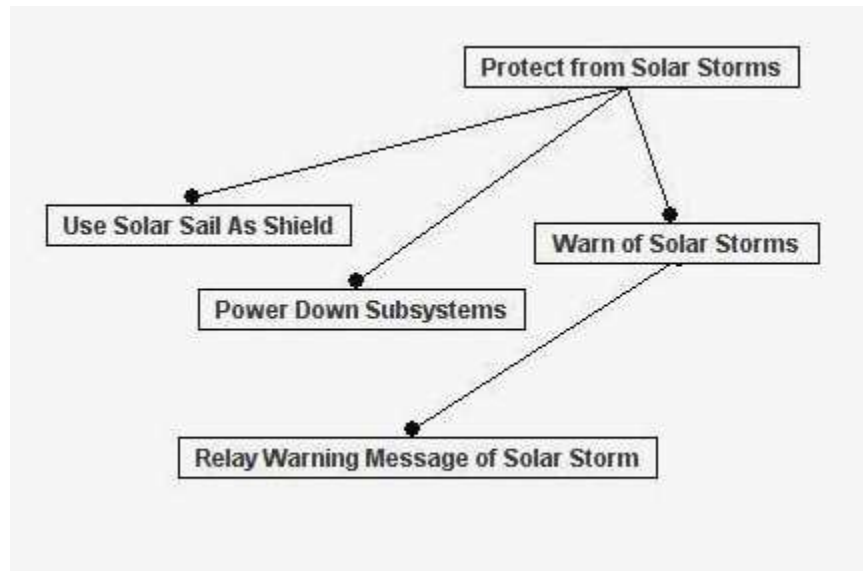
4.2.3 Detailed Design Phase

The Detailed Design Phase of Gaia-PL integrates the application engineering phase of the Weiss and Lai's Family-Oriented Abstraction, Specification and Translation (FAST) product-line methodology [88] with Gaia's Detailed Design Phase [92], [94]. The Detailed Design Phase designs and documents the agents of a multi-agent system product line (MAS-PL) reusing the requirements specification previous developed.

This section describes the development and documentation of an agent of a MAS-PL from the roles and variation points developed in the Analysis and Design Phase, described in Section 4.2.2. We again illustrate this process using the Prospecting Asteroid Mission (PAM) from the requirements specifications schemas developed in the previous section and listed in Appendix D. Note that, since the PAM case study used in this dissertation contains a possibility of 160 unique types of spacecraft (agents), this section and dissertation only illustrates a small set of the possible agents to illustrate Gaia-PL.

4.2.3.1 Designing and Documenting an Agent in Gaia-PL

Upon completion of the initial requirements analysis and development of multi-agent system product line (MAS-PL), it will be necessary to utilize the derived requirements specifications to instantiate a number of members of the system. During this initial deployment of the agents, it is not necessary that all agents be equipped with equal capabilities, intelligence or functionality. Since the prior steps have specified all the possible variation points of the roles in the schemas, we instantiate a new MAS-PL member (i.e., agent) to be added to the MAS-PL system by specifying each new member to be deployed in the Role Deployment Schema. Example Role Deployment Schemas for different configurations of the *SolarStormWarner* role are shown in Figure 24 and Figure 25 for PAM spacecraft with the Feature Model shown in Figure 16 and Figure 23, respectively.



**Figure 23 A Portion of the PAM Feature Model for the SolarStormWarner Role
with only the Passive Variation Point**

Role Deployment Schema: SolarStormWarner		System ID: 2, 3, 8-10
Description:		
Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm. This configuration of the role provides the maximum functionality for this role to monitor, detect and warn of an impending solar storm.		
Variation Points:		
Passive:	The spacecraft does not have the ability to constantly monitor the solar disc to watch for solar storms but can warn other spacecraft after itself receiving a warning message. [SSW-Passive]	
<u>Warm-Spare:</u>	The spacecraft has the ability to constantly monitor the solar disc to watch for solar storms and receive messages from mission control but is acting in a backup/redundant capacity. [SSW-Warm]	
Active:	The spacecraft is tasked to constantly monitor the solar disc and receive warning messages from mission control so that it can warn other spacecraft of an impending solar storm. [SSW-Active]	

Figure 24 Role Deployment Schema for a Configuration of the SolarStormWarner Role

Role Deployment Schema: SolarStormWarner		System ID: 1, 4-7
Description:		
Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm. This configuration of the role provides the minimum functionality for this role to only warn of an impending solar storm.		
Variation Points:		
<u>Passive:</u>	The spacecraft does not have the ability to constantly monitor the solar disc to watch for solar storms but can warn other spacecraft after itself receiving a warning message. [SSW-Passive]	

Figure 25 Role Deployment Schema for a Configuration of the SolarStormWarner Role

The process to design and document an agent of a MAS-PL in the Gaia-PL methodology is as follows:

1. Identify the roles that will constitute the agent to be deployed.
2. For each role identified, create a new Role Deployment Schema and:
 - a. Provide the role's name, unique system(s) identification and a brief description of the role specific to this deployment in the appropriate fields of the Role Deployment Schema. The agent(s) unique identification, to be placed in the System ID field, identifies the specific member(s) of the distributed system to be deployed that has the role configuration described in the particular Role Deployment Schema. For example, if agents with identification numbers 1, 4-7 are to employ the *SolarStormWarner* role in which only variation point Passive is possible (Figure 25), we denote this in the System(s) ID field of the Role Deployment Schema. Similarly, if agents with identification numbers 2, 3, 8-10 are to employ the *SolarStormWarner* role in which the variation points Passive, Warm-Spare and Active are possible (Figure 24), we denote their identification numbers in the System(s) ID field of the Role Deployment Schema. This avoids repetitive manual overhead when designing new members to be deployed in the distributed system and supports traceability, organization and management activities.
 - b. Identify all possible variation points that the role can assume during its lifetime. The set of possible variation points was previously established when the original Role Variation Points Schema was developed for the particular role.

- c. Identify the variation point in which the role will be deployed and specify it in the Role Deployment Schema by underlining it. This variation point represents the default variation point at which the agent will most commonly operate during normal operations. For example, Figure 25 denotes the agents that have the *SolarStormWarner* role in which the variation points *Passive*, *Warm-Spare* and *Active* are possible but where the agent is initially configured to operate at the *Warm-Spare* variation point level.

These steps in Gaia-PL are repeated for all agents that are to be deployed in the MAS-PL. These steps produce a set of completed Role Deployment Schemas describing how different agents of the MAS-PL are to be deployed and how they are initially configured.

4.2.3.2 The Agent Model

We illustrate how an Agent Model, expanded from the Agent Model of Gaia [6], can be derived in this section. The Agent Model graphically illustrates the assignment of roles to agents as well as variation points to roles, similar to that of the Feature Model. The cardinality relationship between agent and role is indicated and all possible variation points are listed for each role. At runtime, the designer annotates the actual cardinality and the specific possible variation points of an agent instance (typically a one-to-one relationship).

In Gaia, the Agent Model defines for each agent the roles that will map to it. Gaia-PL extends this model to additionally map for each role the variation points that may map to it. For example, the partial Agent Model shown in Figure 26 illustrates the *Self-Optimizer*, *Navigator* and *SolarStormWarner* roles used throughout this chapter and their associated variation points. The Agent Model in Gaia-PL will likely be similar to that of the Feature Model and may not be necessary.

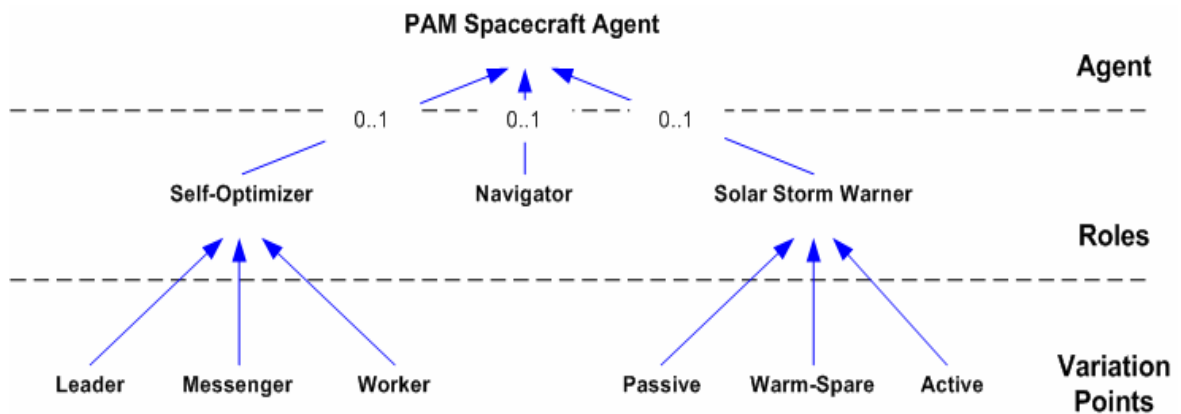


Figure 26 An Excerpt of the Agent Model for the PAM MAS-PL

4.2.4 Summary

The steps of Gaia-PL described in Section 4.2.1 and Section 4.2.2 conform to the domain engineering phase of Weiss and Lai’s Family-Oriented Abstraction, Specification and Translation (FAST) product-line methodology [88] to document the multi-agent system product line’s (MAS-PL) requirements and requirements specifications. The steps of Gaia-PL described in Sections 4.2.3 conform to FAST’s application engineering phase and produce the documentation shown in the detailed design phase shown in Figure 4.

Documenting the requirements specifications in Gaia-PL’s schemas allows easy reuse when instantiating actual agents of a MAS-PL. We detail how the documentation created in this section can easily be reused during both initial development and system evolution using the PAM case study in the next section.

4.3 Requirements Specifications Reuse in the Gaia-PL Methodology

Requirements specification reuse is using previously defined requirements specifications from an earlier system and applying them to a new, slightly different system. Increasing the amount of requirements specification reuse for any given product may reduce the production time and cost of the software system [12].

Requirements specification reuse for multi-agent system product lines (MAS-PL) is simplified in our Gaia-PL methodology by our use of variation points to handle the product-line variabilities in similar systems. The Gaia-PL methodology takes advantage of how the requirements specifications for an agent's role were partitioned and documented in the Role Variation Points Schema and Variation Point Schema based on their variation points.

This section describes how the requirements specifications documentation detailed in Section 4.2 can be reused during the initial deployment of a MAS-PL as well as during its evolution (e.g., the inclusion of new agents, roles or variation points to the MAS-PL).

4.3.1 Reuse During Initial System Development

The members of a distributed system (including a multi-agent system product line (MAS-PL)) often will be heterogeneous in their functional capabilities yet mostly similar in structure. For example, some of the Prospecting Asteroid Mission (PAM) spacecraft may have additional scientific imaging software while others may have additional cluster planning and reconfiguration software.

Heterogeneity may also arise when resources (such as weight limits, memory size, etc.) are limited and different members of a distributed system must assume different roles. In the case of MAS-PLs, agents also may be heterogeneous in terms of their functional capabilities, intelligence levels or other possible variation points (see Section 4.1.2.3). For example, depending on the capability level (e.g., passive, warm-spare or active) of those spacecraft with the *SolarStormWarner* role (see Section 4.2.2.3) among agents, not all agents must support all the possible variation points. That is, not all agents may be capable of monitoring, detecting and warning other spacecraft of an impending solar storm. Rather, most spacecraft may simply be capable of relaying a received

warning to other spacecraft. For this reason, initially deployed members of a MAS-PL will likely contain a role(s) that differs amongst other members in terms of which variation points it is capable of assuming. Several agents of the MAS-PL will have the same role but at different levels of intelligence.

Requirements specification reuse can be exploited during the initial development and deployment of the members of a MAS-PL in Gaia-PL using the Role Deployment Schema, illustrated in Figure 24 and Figure 25. Rather than repeatedly defining the requirements of a role for any given agent (as would be necessary in Gaia), the Role Deployment Schema allows us to define the intelligence levels it can assume. This reuse is possible because the requirements specifications for each of the levels of intelligence were documented in the Variation Point Schemas, and because the agents of a distributed system will be similar.

Thus, to document a particular role for several different heterogeneous members of a distributed system we must only indicate which variation points it can assume and give the reference number(s) to the Role Variation Point Schemas. After assigning variation points to an instance of a role and a role to an instance of an agent, an Agent Model can be used to illustrate an actual instance of an agent, shown in Figure 26. This procedure was described in Section 4.2.3.1.

In the application of the Requirements Documentation and Analysis and Design Phases, described in Sections 4.2.1 and 4.2.2, of Gaia-PL to the PAM case study the 11 Role Variation Point Schemas and 37 Variation Points Schemas (see Appendix D) constructed from the 97 high-level requirements (35 product-line commonality requirements and 62 product-line variability requirements) (see Appendix A, 48 parameters of variation (see Appendix B) and 47 features (see Appendix C) are able to be reused to design and develop 160 unique PAM spacecraft (80 unique *Worker* spacecraft, 48 unique *Leader* spacecraft and 32 unique *Messenger* spacecraft). Thus, the reuse of the

50 schemas developed in Gaia-PL's Analysis and Design Phase were able to accommodate the development of a wide range of PAM spacecraft. A further evaluation of the Gaia-PL approach, compared to that of Gaia, is discussed in Section 4.4.

4.3.2 Reuse During System Evolution

Change is inevitable. Hardware failures or altered mission goals in a deployed distributed system typically necessitate software updates to one or more members. For example, a satellite of the constellation may have a malfunctioning planning and control module that could motivate operators to update that particular satellite's software to erase it and replace it with updated mission planning software. Alternatively, technology or mission goals after the initial deployment of a distributed system routinely evolve in such a way that future deployments of members joining the distributed system will require additional functionality (i.e., new features requiring new requirements).

In the case of the Prospecting Asteroid Mission (PAM), although 1,000 PAM spacecraft will be initially deployed to investigate the asteroid belt, additional spacecraft may have to be deployed if a significant amount of spacecraft are lost due to damage or failures (e.g., solar radiation, collisions, etc.) [71], [77], [83], [84]. The new PAM spacecraft deployed to replace the lost spacecraft may contain additional features not found in previously deployed microsattellites. Examples of the types of evolution additional PAM spacecraft may undergo include improved scientific equipment, new scientific software, new communication devices, new strategies for identifying asteroids of interest, new functionality in existing roles, etc.

A deployed multi-agent system product line (MAS-PL) can evolve in three ways relevant to this work: 1. new agents may be added to the system; 2. new roles with new functionality may be created that future agents can employ; and 3. new variation points may be added to existing roles that future agents can employ. The following subsections

discuss how these types of evolution in a MAS-PL can be accommodated in the Gaia-PL methodology.

4.3.2.1 New Agent MAS-PL Evolution in Gaia-PL

When a MAS-PL evolves, a new agent (i.e., spacecraft) may be deployed to replace a destroyed or failing agent. If this update includes functionality previously defined in the requirements specifications (Role Variation Points Schema and Variation Point Schemas), it suffices to modify the Role Deployment Schema and, possibly, the Agent Model to reflect the update.

If the evolution of the MAS-PL involves a new agent to be deployed that includes additional functionality not previously defined in the requirements specifications (Role Variation Points Schema and Variation Point Schemas), updates to the MAS-PL's requirements specifications is needed. The requirements specifications patterns detailed in Sections 4.2.1 and 4.2.2 are extensible in that it can accommodate this kind of system evolution by being able to include a new set of requirements while still reusing the previously documented requirements. This situation is discussed in Section 4.3.2.2 and Section 4.3.2.3.

4.3.2.2 New Role MAS-PL Evolution in Gaia-PL

The addition of a new role during evolution within a multi-agent system product line (MAS-PL) is analogous to the inclusion of a role during initial system development, as described in Section 4.2. Briefly, we create a new Role Variation Points Schema and a Variation Point Schema(s) just as during the initial development of a MAS-PL. Following the creation of a Role Variation Points Schema and a set of Variation Point Schemas, the process in Gaia-PL's Detailed Design Phase, outlined in Section 4.2.3, is used to instantiate a new agent with the new role.

Note that a new role should have new requirements and/or features associated with it. The new requirements/features that are implemented in the functionality of a role should additionally be represented in the Commonality and Variability Analysis (see Section 4.2.1.1), the Parameters of Variation table (see Section 4.2.1.1) if it is a new product-line variability requirement and the Feature Model (see Section 4.2.1.3). The inclusion of new requirements to a MAS-PL can be handled as is traditionally done for the evolution of a software product line. This process is described in [12], [67], [88] and is not the focus of this research. However, the use of DECIMAL [23], [58], [59] (see Section 4.2.1.2) in Gaia-PL may ease the inclusion of new requirements as a result of new roles being added to the MAS-PL because of its ability to automatically verify that the new role and new agents abide by the MAS-PL's constraints.

4.3.2.3 New Variation Point MAS-PL Evolution in Gaia-PL

The addition of a new variation point to an existing role during multi-agent system product line (MAS-PL) evolution, however, requires a modification to existing Role Variation Points Schema documentation as well as the creation of a new Variation Point Schema. To describe and illustrate the process of updating Gaia-PL's requirements specifications in the event that a new variation point must be added to an existing role as a result of evolution, we use the following hypothetical situation in the Prospecting Asteroid Mission (PAM) case study as motivation:

After the initial deployment of the PAM spacecraft, mission engineers discover that, in addition to the *Leader*, *Messenger* and *Worker* types of spacecraft already present in the PAM swarm, an additional *Scout* type of spacecraft is desired to better investigate the asteroid belt. The *Scout* spacecraft would be tasked with working mostly independently to quickly survey asteroids, assess their relevance to the mission goals and decide which asteroids should be

further explored by a PAM subswarm consisting of *Leaders*, *Messengers* and *Workers*.

Thus, the new *Scout* type of PAM spacecraft will include some of the functionality of the *Leader* and *Worker* spacecraft but will additionally include new functionality. The inclusion of this new type of spacecraft to the PAM MAS-PL will necessitate the updating of portions of the requirements specifications. One such update needed is to include a new “Scout” variation point to the *Self-Optimizer* role, described in Sections 4.2.2.1 and 4.2.2.3, to include the functionality that provides the *Scout* spacecraft with the ability to optimize itself in order to better satisfy its scientific goals.

To accommodate a new variation point in an existing role for the use in future deployments of the MAS-PL, as described in the above scenario, using the Gaia-PL methodology, the following process suffices:

1. Update the Role Variation Points Schema to which the new variation point corresponds, and add the new variation point, along with a description and schema reference identification, to the Variation Points section. An example of this from the scenario described above for the *Self-Optimizer* role is shown in Figure 27. Note that the original Role Variation Points Schema for the *Self-Optimizer* role is given in Figure 11. Figure 27 expands the Role Variation Points Schema, from Figure 11 to include the new Scout variation point.
2. Create a new Variation Point Schema, shown in Figure 28 for the new variation point giving the role's name, variation point's name and a unique variation point identifier in the appropriate fields.

Role Variation Points Schema: SelfOptimizer		Schemata ID: SO
Parameters of Variation: P4, P5, P6		
Description:		
At the swarm-level, the collection of these roles within all the spacecraft aid in autonomously and continuously improving the spacecraft's ability to identify, explore and communicate the information discovered while investigating asteroids. At the spacecraft-level, these roles aid in the spacecraft to continuously learn and improve its specialized abilities and communicate its findings with other similar spacecraft.		
Variation Points:		
<u>Core:</u>	The core elements of a spacecraft to be able to optimize itself in regards to general spacecraft functions so that it can continuously learn from the environment and perform better within the swarm. [SO-Core]	
Leader:	The elements needed in a <i>leader</i> spacecraft to be able to optimize itself in regards to its ability to best manage, oversee and direct the swarm to optimize the swarm's ability to achieve scientific goals. [SO-Leader]	
Messenger:	The elements needed in a <i>messenger</i> spacecraft to be able to optimize itself in regards to its ability to best perform the communication necessary within the swarm so that commands and information can best be transmitted. [SO-Messenger]	
Scout:	The elements needed in a <i>scout</i> spacecraft to be able to optimize itself in regards to its ability to independently survey asteroids and decide which asteroids should be further investigated by a PAM subswarm. [SO-Scout]	
Worker:	The elements needed in a <i>worker</i> spacecraft to be able to optimize itself in regards to its ability to best optimize its ability to achieve its own scientific goals. [SO-Worker]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time. However, a spacecraft that may switch is operating variation point (i.e., P2=True or P3=True) may have this variation point alter at runtime.		

Figure 27 Updated Role Variation Points Schema for the Self-Optimizer Role as a Result of Evolution

3. Provide any variation points that the new variation point must inherit. Additionally denote the associated requirements and parameters of variation in the appropriate fields of the new Variation Point Schema.
4. Document the variation point indicating how the new variation point differs from previously defined variation points in the Description section.

Role Schema: SelfOptimizer	Schema ID: SO-Scout
Variation Point: Scout	
Inherits: SO-Core	
Requirements: N/A	
Parameters of Variation: N/A	
Description: The elements needed in a <i>scout</i> spacecraft to be able to optimize itself in regards to its ability to independently survey asteroids and decide which asteroids should be further investigated by a PAM subswarm.	
Activities and Protocols: Calculate EvaluateAsteroidStrategy, <u>SendNewAsteroidData</u> ,	
Permissions:	
Reads -	
<i>currentPosition</i>	// current position of the spacecraft in the // asteroid belt
<i>messengerVector</i>	// vector of nearby <i>messenger</i> spacecraft // to aid in sharing optimization information
<i>scienceGoal</i>	// current scientific goal pursued by the // spacecraft
Changes -	
<i>asteroidEvaluationStrategy</i>	// strategy for spacecraft's approach // in surveying an asteroid
<i>asteroidIdRules</i>	// vector of rules that is used to identify // asteroids of interest given preliminary // data points on the asteroid
<i>surveyedAsteroidHistory</i>	// history log of the asteroids surveyed by // the scout spacecraft
Generates -	
<i>asteroidMap</i>	// rough map of the asteroids surveyed // that have yet to be further explored
<i>newSurveyRule</i>	// new rule devised by the role to use when // surveying and evaluating an asteroid
<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information
<i>sciExplorationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // achieve its scientific goals
Responsibilities:	
Liveness - If the spacecraft is functioning properly, the role will eventually improve its ability to independently survey and identify asteroid of interest that should be further explored by a PAM subswarm.	
Safety - None.	

Figure 28 The New Variation Points Schema for the Scout Variation Point of the Self-Optimizer Role as a Result of Evolution

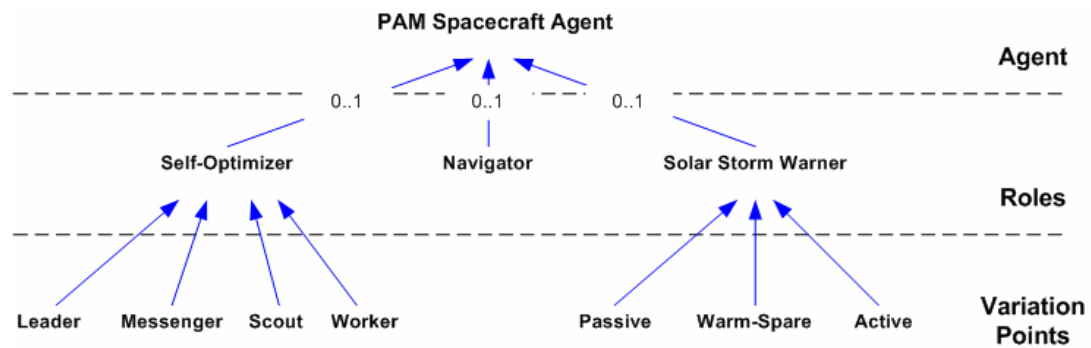


Figure 29 Excerpt of the Updated Agent Model to Reflect the Addition of the Scout Variation Point to the Self-Optimizer Role

5. Identify the protocols, activities, permissions and responsibilities that are particular to only that variation point. That is, define the protocols, activities, permissions and responsibilities that are not found in any of the lower intelligence level variation points and that are not found in any other variation points.
6. Document and define the identified protocols, activities, permissions and responsibilities in the appropriate sections of the Variation Point Schema.
7. Update the Agent Model(s) to reflect the inclusion of the new variation point for the role. The new Scout variation point for *Self-Optimizer* role is included in the updated Agent Model in Figure 29.

These steps will produce a new variation point for a role and the accompanying Variation Point Schema for future versions of members of the system.

4.4 Evaluation of the Gaia-PL Methodology

This section evaluates the Gaia-PL methodology in the context of its application to the Prospecting Asteroid Mission (PAM) case study. We also provide a comparison of the Gaia-PL and Gaia methodologies in the context of the PAM case study and a brief discussion of the results.

4.4.1 The PAM Case Study

The application of the Gaia-PL methodology to the Prospecting Asteroid Mission (PAM) during the Requirements Documentation Phase documented 97 high-level product-line requirements in the Commonality and Variability Analysis (CVA), discussed in Section 4.2.1.1 and shown in Appendix A. The 97 high-level product-line requirements included 35 commonality requirements and 62 variability requirements. Thus, we found that approximately one-third of the requirements of the PAM case study were in all spacecraft regardless of its specialized role (i.e., a leader, messenger or worker designated spacecraft). The 62 variability requirements were analyzed and grouped into 48 parameters of variation, discussed in Section 4.2.1.1 and shown in Appendix B. Further, the product-line requirements of the PAM case study were partitioned into 47 features, discussed in Section 4.2.1.3 and shown in Appendix C.

In the Analysis and Design Phase of Gaia-PL, we identified 13 unique roles for the PAM case study that were documented in 2 Role Schemas, 11 Role Variation Points Schemas and 39 Variation Point Schemas, as discussed in Section 4.2.2 and shown in Appendix D. These requirements specifications schemas can be used to design and develop 160 unique PAM spacecraft (80 unique *Worker* spacecraft, 48 unique *Leader* spacecraft and 32 unique *Messenger* spacecraft). Thus, the reuse of the 52 schemas developed in Gaia-PL's Analysis and Design Phase was able to accommodate the development of a wide range of PAM spacecraft.

To measure the impact and ability of the inclusion of variation points into the roles of an agent in a MAS, this evaluation measured the number of variation points defined for each role and the number of parameters of variation and requirements implemented in each variation point. These measurements provide an insight into the extent of the variable behavior of an agent that can be defined for a role and partly

illustrates the advantage of the inclusion of product-line engineering into the development of a MAS in Gaia-PL.

The Role Variation Points Schemas developed for the PAM case study during the Analysis and Design Phase of Gaia-PL had an average of 3.9 variation points where the minimum number of variation points identified for a role was 2 (e.g., the *LeaderPlanner* role, see Appendix D, page 264), and the maximum number of variation points identified for a role was 10 (e.g., the *Worker* role, see Appendix D, page 271). Additionally, the Role Variation Points Schemas represented an average of 4.8 of the parameters of variation (see Appendix B) where the minimum number of parameters of variation identified for a role was 1 (e.g., the *WorkerCooperation* role, see Appendix D, page 282), and the maximum number of parameters of variation identified for a role was 21 (e.g., the *Worker* role, see Appendix D, page 271).

Further, the Role Variation Points Schemas had implemented an average of 4.1 high-level requirements from the CVA where the minimum number of requirements implemented in a variation point was 1 (e.g., the NIRSspec variation point of the *Worker* role, see Appendix D, page 274), and the maximum number of requirements implemented in a variation point was 14 (e.g., the Core variation point of the *Self-Coordinator* role, see Appendix D, page 255). Note that many of the high-level requirements were implemented in several roles (i.e., were cross-cutting in more than one role). For example, requirement C_M4 “Every spacecraft shall be able to know its current position” is needed in multiple roles.

Of the 11 Role Variation Points Schemas identified for the PAM case study, 8 contained a variation point that must be included if the role is included in the agent. For example, the *Messenger* role (i.e., a role that not every agent will contain, see the Feature Mode in Figure 9), shown in Appendix D, page 288, contains two variation points one of which is required (i.e., the “Core” variation point). For the *Messenger* role, the “Core”

variation point captures 6 of the 8 requirements that are associated with the functionality possible in the *Messenger* role. That is, 6 of the 8 of the requirements were common to all agents containing the *Messenger* role while only 2 of the 8 requirements were variable functionality.

The *SolarStormWarner* role, discussed (see Section 4.2.2.3) similarly captured a large portion of the role's common requirements in its required variation point. However, unlike the *Messenger* role, the *SolarStormWarner* role (see Appendix D, page 296) is required for all PAM spacecraft (see the Feature Mode in Figure 9). Nevertheless, the common variation point for the *SolarStormWarner* role captured 54.5% of the common requirements in its Variation Point Schema (see Appendix D, page 297).

Among the 8 Role Variation Points Schemas of the PAM case study that contained a variation point that must be included if the role is included in the agent, an average of 41% of the requirements were found to be common to the required variation point of the role. The minimum amount of common requirements for a role was 13% for the *Worker* role (see Appendix D, page 271) and the maximum was 75% for the *Messenger* role, described above. Thus, using the Role Variation Points Schema in Gaia-PL captures, at least in the case of the PAM case study, a portion of the requirements that are common to all agents with a particular role and can be reused to develop agents with the any allowable combination of the role's variation points. Further, the ability to separately capture the common requirements of a role in a variation point avoids the need to have the common requirements repeated in several role schemas for each of the variation points, as would be needed using the Gaia methodology (discussed in the next section).

The design and documentation of the 39 Role Variation Point Schemas for the PAM case study took approximately 30 minutes each for a total of 19.5 hours. Thus, for each requirement implemented in a Role Variation Point Schema, it was found in this

case study that an average of 7.3 minutes was needed to document the requirement's specification in a Variation Point Schema.

4.4.2 Comparison to the Gaia Methodology

The contribution of the Gaia-PL methodology detailed in this dissertation is to provide a way to develop software engineering assets that can be readily reused to build the agents of a multi-agent system product line (MAS-PL). As mentioned in the previous section, the application of Gaia-PL to the Prospecting Asteroid Mission (PAM) case study used in this dissertation yielded 39 Variation Point Schemas that can be reused to build 160 unique agents (i.e., spacecraft) of the PAM swarm.

The mechanism to provide the reusable assets in the Gaia-PL methodology centers on the identification and separation of the commonalities of the agents and the agent's roles and the refinement of the variabilities of the agents and the agent's roles in separate software engineering artifacts. The ability to separately capture the common requirements of a role in a variation point avoids the need to have the common requirements repeated in several role schemas for each of the variation points. In this section, we discuss this advantage of Gaia-PL by a comparison to the application of Gaia to the PAM case study.

The application of the Gaia methodology from the requirements for the PAM case study listed in the Commonality and Variability Analysis (CVA) given in Appendix A, yields 48 (a 19% increase compared to Gaia-PL) of Gaia's Role Schemas (similar to Gaia-PL's Variation Point Schemas) to document the same requirements specifications. To accommodate the requirements of the PAM case study, Gaia needs to implement a role for each of our variation points where the variable variation points (i.e., non-required) variation points additionally including the required variation point functionality.

For example, in the *SolarStormWarner* role, described in Section 4.2.2.3 and given in Appendix D, page 296, a new Role Schema in Gaia has to be created for the Passive, Warm-Spare and Active variation points. In addition, the new roles for the Warm-Spare and Active roles also has to include the functionality (i.e., requirements) of the Passive variation point. Thus, the Gaia Role Schema for the Warm-Spare and Active *SolarStromWarner* roles combine and repeat the functionality of the Passive variation point.

Although this approach using Gaia accommodates the functionality of the PAM case study, it does not clearly document an agent's ability to change from one set of functionality of a role to another (e.g., from the Warm-Spare to the Active functionality of the *SolarStormWarner* role). Rather, Gaia has to combine the functionality from the variation points into a single role. Yet, this does not keep the modularity of the differing types of functionality in a role as in Gaia-PL and may confuse developers during coding. Secondly, the non-hierarchical nature of Gaia is not able to provide any linking relationships between related roles (e.g., from the Warm-Spare to the Passive functionality of the *SolarStormWarner* role) as in Gaia-PL. Lastly, some of the functionality will be unnecessarily repeated (e.g., the Passive functionality also must be included in the Warm-Spare and Active roles of a *SolarStormWarner*).

Although the application of the Gaia methodology to the PAM case study only increases the number of schemas needed by approximately 19% compared to our Gaia-PL approach, the number of redundantly implemented requirements further illustrates the advantage of Gaia-PL. As discussed in Section 4.4.1, we found that an average of 41% of the requirements implemented in the required variation points of 8 of the 11 Role Variation Points schemas were common to all variation points of the role. However, since the redundant requirements need to be documented for each variation point to create a new role in Gaia, the set of Role Schemas has a 66.5% rate of requirements that have

already be documented in another role. Of these 8 roles identified in Gaia-PL (with 35 variation points), Gaia created 41 roles that contained 33 redundant requirements. Due to the high number of redundant requirements, the 33 Role Schemas created in Gaia documented 222 requirements (of which 66.5% or 147 requirements are redundant). Assuming that it continues to take an average of 7.3 minutes per requirements to document in a requirements specification, the Gaia approach incurred an additional 17.8 hours to derive and document compared to the Gaia-PL approach discussed in this chapter.

4.4.3 Discussion

The evaluation of our Gaia-PL methodology using the Prospecting Asteroid Mission (PAM) case study measures Gaia-PL's ability to capture the common parts of a multi-agent system product line (MAS-PL) so that they can be reused along with the variable parts to design and develop an agents. It was shown in the previous section that compared to Gaia, an Agent-Oriented Software Engineering (AOSE) methodology that does not explicitly partition the common and variable parts of a MAS-PL, Gaia-PL's ability to reuse the common parts of an agent's role reduces the work and time required to design and develop an agent. However, the evaluation of our Gaia-PL methodology does not come without caveats. In this section, we discuss some of the caveats of our evaluation.

The PAM case study used in this dissertation had requirements that fit nicely into adopting a software product-line engineering approach. The requirements gathered for the PAM case study readily fit into a Commonality and Variability Analysis (CVA) because the common and variable functionalities of the spacecraft were clear. A characteristic of the PAM case study aiding its adoption into a product line approach was its basis on the Autonomous Nano-Technology Swarm (ANTS) concepts. The

requirement that all PAM spacecraft implement the concepts of the ANTS mission provided a natural mechanism to define the commonalities. In addition, the variability requirements of the PAM mission partly focused on the differing functionality of the different types of spacecraft (i.e., Leader, Messenger or Worker). Further, there was approximately a 2:1 ratio of variable requirements to commonality requirements. These factors certainly contributed to the clear advantage that Gaia-PL compared to Gaia. However, for MAS-PL's with less variability and mostly common functionality among a role, or for MAS-PL's in which no variation points for a role can be identified, Gaia-PL will not provide such clear advantages.

In the case where a MAS-PL has little variability, although Gaia-PL will not necessarily provide the advantage in reuse described in Section 4.3, it will not incur enough overhead to be a disadvantageous approach compared to Gaia. Unlike Gaia, Gaia-PL does require the documentation of variation points (if any) in a Role Variation Points Schema which will incur additional development time. Yet, for MAS-PL's that will have few variation points (and thus few variabilities), the Role Variation Points needed will be few and require very little development time. Thus, the Gaia-PL approach would still provide some advantage for those roles which have variation points while not incurring a large overhead.

This evaluation did not consider design alternatives in our application of Gaia-PL to the PAM case study. That is, we did not design and evaluate different ways of defining a role's variation points nor did we design and evaluate different ways of defining the roles possible in an agent. Thus, the results obtained from our evaluation might differ if we had used a different design alternative for the PAM case study.

Although the evaluation of Gaia-PL was performed on a relatively large MAS-PL case study, the results only report the performance of Gaia-PL on a single case study. The application of Gaia-PL on different MAS-PL application, in particular applications with a

different profile of variable functionality, will yield different results. Thus, the evaluation reported in this dissertation should only serve as a proof-of-concept measurement rather than the results that should be expected in its use on other MAS-PL applications.

Despite these caveats, the evaluation of Gaia-PL indicates its advantages in designing, developing and documenting MAS-PLs that have some degree of variability. Gaia-PL's ability to hierarchically define the roles of an agent, capture the common and variable functionality of an agent and reuse the common functionality of a role to design and develop a wide-range of agents of the MAS-PL recommends its use. In particular, the use of Gaia-PL as an extension of Gaia allows the software developer to take advantage of the reuse potential in Gaia-PL along with the other models, abstractions and analysis tools of Gaia to provide the mechanisms to efficiently design and develop a MAS-PL.

4.5 Summary

This chapter detailed the design and development of a multi-agent system product line (MAS-PL) using our Gaia-PL methodology. The Gaia-PL methodology produces reusable software engineering assets so that building systems of the MAS-PL can be done efficiently and with a high-degree of reuse. Software product-line engineering concepts were integrated into agent-oriented software engineering (AOSE) by identifying, defining and using *variation points* to build a MAS-PL. We illustrated the documentation of MAS-PL requirements in a Commonality and Variability Analysis and a Parameters of Variation table and detailed the documentation of requirement specifications in Gaia-PL's schemas. These schemas partitioned the commonality requirements and variability requirements into separate schemas for specific roles using a Feature Model as a guide. Gaia-PL's schemas were then shown to be reused to build specific types of agents for a MAS-PL.

This chapter discussed and illustrated the reuse of the requirements specifications during initial system development of a MAS-PL as well as during system evolution. To highlight the advantages of Gaia-PL, we differentiated our methodology from previous work by illustrating Gaia-PL's ability to capture reuse and avoid the redundant work and increased development cost (i.e., additional time) needed to develop the agents required as done in previous MAS work. Finally, an evaluation of our Gaia-PL methodology on the PAM case study illustrated the development cost savings and other advantages of our approach.

For safety-critical MAS-PLs, the Gaia-PL methodology described in this chapter provides no mechanisms to ensure that the MAS-PL being built is indeed safe. Chapter 5 builds upon the Gaia-PL methodology by detailing safety analysis techniques and tools for the analysis of safety-critical MAS-PLs in the context of Gaia-PL.

CHAPTER 5. SAFETY ANALYSIS FOR SAFETY-CRITICAL MULTI-AGENT SYSTEM PRODUCT LINES³

Chapter 4 detailed our Agent-Oriented Software Engineering (AOSE) methodology, Gaia-PL (Gaia – Product Line) for developing reusable requirement specifications for a multi-agent system product line (MAS-PL) and then reusing them for initial system development as well as during evolution. This chapter focuses on the development of reusable safety analysis artifacts for safety-critical MAS-PL in the context of our Gaia-PL methodology. The goal is to develop reusable safety analysis artifacts for a MAS-PL. This chapter describes the product-line safety analysis techniques and tools we have developed and adapted for the use during the design and development of safety-critical MAS-PLs. The safety analysis techniques and tools described in this chapter aim to provide some assurance that core assets defined in the domain engineering phase are being safely reused during the application engineering phase. We again use the Prospecting Asteroid Mission (PAM) case study, described in Chapter 3, to illustrate and evaluate our safety analysis techniques and tools.

³ This chapter extends our previous work that has appeared in papers at *2004 High Assurance Systems Engineering Conference (HASE'04)*, *2005 International Symposium on Software Reliability Engineering (ISSRE'05)*, *2005 International Conference on Software Engineering's Workshop on Software Engineering for Large-Scale, Multi-Agent Systems (SELMAS'05)*, *2006 Workshop on Innovative Techniques for Certification of Embedded Systems (ITCES'06)*, and *Automated Software Engineering Journal*, 2006 all co-authored with Robyn R. Lutz; and *2007 International Conference on Software Engineering (ICSE'07)*, co-authored with Meredith Humphrey, Lada Suvorov, Prasanna Padmanabhan and Robyn R. Lutz.

5.1 Software Safety Analysis for Multi-Agent System Product Lines

This section examines the need for safety analysis techniques and tools for multi-agent system product lines (MAS-PL) and provides an overview of the safety analysis techniques and tools for MAS-PLs detailed in this chapter.

5.1.1 The Need for Safety Analysis for Developing Multi-Agent Systems

Multi-agent systems (MAS), like other software systems may be safety-critical. A safety-critical system is a system that can directly or indirectly compromise safety by placing a system in to a hazardous state causing the potential loss or damage of life, property, information, mission or environment [44]. Thus, although the Prospecting Asteroid Mission (PAM) case study used in this dissertation may not directly cause the loss of human life, failures in the PAM spacecraft can result in the loss or damage of property (i.e., the spacecraft), information (i.e., the data gathered on the asteroid belt) and/or mission. Thus, the PAM case study, and similar agent-based systems, necessitates safety analysis to ensure that no undesirable behaviors will occur that may compromise the system's mission. Further, for some agent-based systems, safety certification may be required.

However, the safety analysis of a MAS presents challenges not found in other software systems. In particular, one of the most challenging characteristics of a MAS preventing the use of traditional software safety analysis techniques and tools is that it is difficult to verify that the emergent behavior of such systems will be proper and that no undesirable behaviors will occur. Although the emergent properties of a distributed, MAS make the systems more powerful and adaptable, they are inherently more difficult to design and provide assurance that the proper, safe behaviors will emerge. In addition, the

complexity of distributed MAS, such as PAM, in their ability to interact with each other and dynamically alter their functionality further complicates the safety analysis of such systems. Unless safety analysis techniques and tools, along with further validation and verification techniques can assure the correct, safe behavior and interactions of a MAS, the safety of such software systems can not be assured.

For MAS that consist of a high number of similar yet slightly different agents, as in the PAM case study, a product-line safety analysis approach is advantageous. Like the product-line approach described in Chapter 4, a product-line approach to safety analysis allows the reuse of portions of the safety analysis for multiple agents of the multi-agent system product line (MAS-PL). The ability to reuse portions of the safety analysis for a new agent can significantly reduce the burden of safety analysis of the entire system. Further, reusable safety analysis assets can be used to make a safety case for the software during the system certification, can aid in verifying the safety requirements of the system and can discovering safety requirements missed in the initial requirements specification.

Certification is a process whereby a certification authority determines if an applicant provides sufficient evidence concerning the means of production of a candidate product and the characteristics of the candidate product so that the requirements of the certifying authority are fulfilled [31], [40], [69], [72]. Software safety analysis techniques, similar to those detailed in this chapter, have previously been shown to contribute to the certification of software-intensive systems in [2], [55]. However, little work has been specifically aimed at MAS-PLs.

Certification may apply to the development process, the developer or the actual product [55]. Since it is insufficient to certify the process or developer for the software of safety-critical systems, building a safety case that provides “an argument accompanied by evidence that all safety concerns and risks have been correctly identified and mitigated” [26] aids in the certification of the product. The safety analysis techniques

and tools described in this chapter integrate the reuse potential of safety analysis assets into the design and development of MAS-PL so that they can be used to better make a safety case when system certification is required as well as allowing the safety engineer to verify the safety requirements of the system and can discover missing safety requirements. These safety analyses provide some assurance that core assets defined in the domain engineering phase are being safely reused during the application engineering phase.

The safety analysis techniques and tools presented in this chapter provide safety analysis techniques for a safety-critical MAS-PL in the context of our Gaia-PL methodology. In the following section we provide an overview of our safety analysis techniques and tools in the context of Gaia-PL.

5.1.2 Overview of Our Safety Analysis Techniques for Developing Multi-Agent System Product Lines

Figure 30 provides an overview of the safety analysis techniques that we have developed and adopted for the use in designing and developing safety-critical, multi-agent system product lines (MAS-PL) in the context of our Gaia-PL methodology. To provide reusable safety analysis assets for the design and development of safety-critical MAS-PL, an extended Bi-Directional Safety Analysis (BDSA) approach [32], [54], [55] was used. BDSA combines a search from potential failure modes to their effects with a search from possible hazards to the contributing causes of each hazard. The use of a BDSA approach requires the use of forward and backward safety analyses. In the work described in this chapter, we use Software Failure Modes, Effects and Criticality Analysis (SFMECA) and Software Fault Tree Analysis (SFTA) as the forward and backward search technique, respectively.

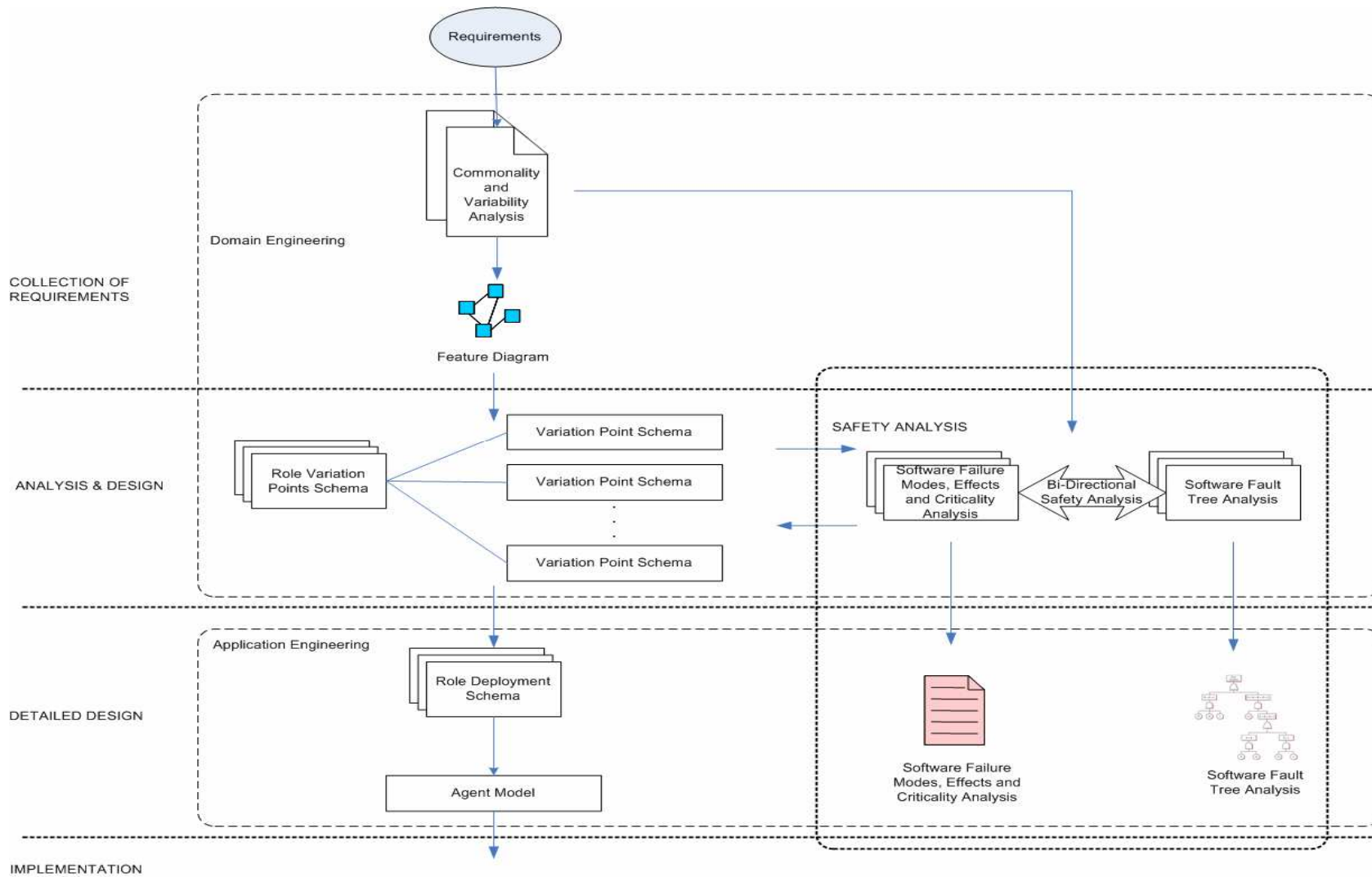


Figure 30 An Overview of the Safety Analyses for MAS-PL in the Gaia-PL Methodology

The extended SFMECA safety analysis technique presented in Section 5.2 provides a systematic process to derive a forward-based safety analysis asset from the Variation Point Schemas of a role in our Gaia-PL methodology (see Chapter 4). The SFMECA tables derived using this approach are directly associated to the variation point of a role and can be reused for an agent with the specific role and variation point.

The SFTA, discussed in Section 5.3, presents our technique that cleanly extends SFTA to software product lines. This product-line SFTA (PL-SFTA) can be constructed for an entire product line and product-line members' fault trees can be derived from the PL-SFTA. PLFaultCAT, a graphical tool to construct a product-line SFTA, supports this technique and then allows users to automatically derive a product-line members' fault tree given the variabilities to be included.

BDSA, discussed in Section 5.4, is then used to verify the completeness of the forward and backward search techniques. The forward and backward techniques can be viewed as complementary since the output of the forward technique (i.e., the potential system-wide hazards) should match-up with the inputs of the backward technique. Similarly, the output of the backward technique (i.e., the low-level, local errors that cause a system-wide hazard) should match-up with the inputs of the forward technique. Thus, the BDSA can discover the missing safety requirements can be derived from the SFMECA and SFTA safety analysis assets and can assist in verifying the adequacy of the existing safety requirements and design. The resulting MAS-PL safety assets and verification aid in efficiently assembling a safety case during system certification.

The remainder of this chapter details each of these safety analysis techniques and tools and illustrates them using the PAM MAS-PL case study.

5.2 Software Failure Modes, Effects and Criticality Analysis for the Gaia-PL Methodology

The forward analysis technique used in this work for the safety analysis of a safety-critical, multi-agent system product line (MAS-PL) is a Software Failure Modes Effects and Criticality Analysis (SFMECA). To accommodate the design and development of a MAS-PL in Gaia-PL, we have adapted the SFMECA technique to analyze and define a SFMECA tailored to the variation points in our Gaia-PL methodology to produce a safety analysis technique specific to MAS-PL.

In our Gaia-PL methodology, the requirements specifications of the variation points of a role are document in a Variation Point Schema (see Section 4.2.2.3). The Variation Point Schema conveniently partitions a role’s requirements specifications into events (functionality) that the role can perform and data that the role can access and generate. In Gaia-PL’s Variation Point Schema, the events that a role or variation point can perform are the non-underlined methods listed in “Activities and Protocols” section, and the data that the role or variation point can access and generate are listed in the “Permissions” section. For example, in the *CollisionProtector* Variation Point Schema of the Prospecting Asteroid Mission (PAM) case study used in this dissertation, shown in Figure 31, the events that this role can perform include *Analyze3DModel*, *DetectNearbySpacecraft*, etc. Similarly, the data that this role can access and generate include, *position*, *velocityIncrement*, etc.

The SFMECA tables created in this work are specific to a variation point. For example, a separate SFMECA table will be created for the *CollisionProtector* Variation Point, shown in Figure 31, so that this safety analysis can be readily reused for all agents with the *CollisionProtector* Variation Point. Additionally, like [32] we partition the SFMECA into separate analyses on the data and events. We use guidewords of [54] to steer our investigation into the possible failures within a MAS-PL.

Role Schema: CollisionProtector	Schema ID: CP
Variation Point: CollisionProtector	
Inherits: None	
Parameters of Variation: N/A	
Requirements: C_SP1, C_SP2, C_SP3, C_SP4, C_SP5	
Description: Provides the spacecraft with the functionality to autonomously protect itself from colliding with other spacecraft and nearby asteroids.	
Activities and Protocols: Analyze3DModel, DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids, MoveToAvoidCollision, AcceptAsteroid3DModel, AcceptCurrentPosition, AcceptCurrentTrajectory, AcceptSpacecraftLocations, NegotiateCollisionAvoidance, PingNearbySpacecraft, RequestAsteroidPositions, RequestCurrentPosition, RequestCurrentTrajectory, RequestSpacecraftLocations	
Permissions:	
Reads -	
<i>curScienceGoalFactor</i>	// current spacecraft scientific goal factor
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
supplied <i>asteroid3DModel</i>	// 3D model of an asteroid supplied
supplied <i>asteroidPositions</i>	// positions of nearby asteroids
supplied <i>subswarmVector</i>	// vector of nearby spacecraft positions
supplied <i>spacecraftPos</i>	// current position of a nearby spacecraft
	// supplied by a <i>messenger</i> or <i>leader</i>
supplied <i>spacecraftTraj</i>	// current trajectory of a nearby spacecraft
	// supplied by a <i>messenger</i> or <i>leader</i>
Changes -	
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
Generates -	
<i>collisionRiskFactor</i>	// derived risk to spacecraft for an // impending collision
<i>riskToGoalFactor</i>	// calculated value of the current risk factor // to the advantage of pursuing scientific // exploration
<i>nearbyAsteroids</i>	// vector of nearby asteroids that must be // avoided to prevent a collision
<i>nearbySpacecraft</i>	// vector of nearby spacecraft that must be // avoided to prevent a collision
Responsibilities:	
Liveness -	
None.	
Safety -	
Prevent the collision with other spacecraft and nearby asteroids.	

Figure 31 The Variation Point Schema for the CollisionProtector Role

Each activity of a variation point (the non-underlined keywords listed in Gaia-PL’s Variation Point Schemas under the “Protocols and Activities” section) is essentially an event (i.e., some functionality) that the variation point can execute. To construct a SFMECA table for the events that a role’s variation point can execute, as in a standard SFMECA we use the following keywords to guide our analysis: “halt/abnormal termination”, “omission”, “incorrect logic/event” and “timing/order”.

Similarly, constructing the SFMECA data table using Gaia-PL’s Variation Point Schemas, the “Permissions” section lists each datum that the role or variation point can access, alter or generate. To construct a SFMECA table for the data that a role uses, we use the following keywords to guide our analysis: “incorrect value”, “absent value”, “wrong timing” and “duplicated value”.

Since a role definition depends on its variation point(s) in the Variation Point Schemas of a role, detailed in full in Section 4.2.2.2, the derived SFMECA captures the possible event and data failures for all the near-identical agents.

In Section 5.2.1 we describe the construction of the SFMECA event table for a Variation Point Schema and then in Section 5.2.2 the construction of the SFMECA data table for the *CollisionProtector* variation point, shown in Figure 31. Note that the SFMECA creation process described here occurs during the domain engineering phase of Weiss and Lai’s Family-Oriented Abstraction, Specification, and Translation (FAST) model. Thus, the SFMECA table represents the possible failures for the entire set of products in the MAS-PL.

The *CollisionProtector* variation point is tasked with preventing the spacecraft from colliding with other spacecraft and nearby asteroids. The failure of this variation point may lead to the collision and loss of one or more spacecraft and thus warrants safety analysis. We include a portion of the SFMECA for the *CollisionProtector* Variation Point in Table 3, Table 4 and Table 5 as well as in Appendix E.

5.2.1 Constructing a SFMECA Event Table for a Variation Point in Gaia-PL

The procedure to construct a SFMECA table for the events from Gaia-PL's Variation Point Schema(s) using the event guidewords consists of the following steps:

1. For each role:
 - a. Create a new SFMECA event table similar to that shown in Table 3 and fill in the role's name and its possible variation points.
 - b. Then, for each of the variation points possible, listed in the role's Role Variation Points Schema (see Section 4.2.2.2):
 - i. For each activity listed in the *Protocols and Activities* section of the Variation Point Schema:
 - c. Provide the event name in the "Event" column.
 - c. Apply each of the failure mode keywords (i.e., "halt/abnormal termination", "omission", "incorrect logic/event" and "timing/order") to the event. For each keyword:
 - i. Provide the event failure mode keyword in the "Failure Mode" column.
 - ii. Describe the possible local effect(s) if the keyword failure happened to the event under consideration in the "Local Effect(s)" column. The local effect will likely only affect this role or this agent and its description should not include the propagation of its failure to other agents or components of the global system.
 - iii. Describe the possible system-level effect(s) if the keyword failure mode occurred in the "System Effect(s)"

column. This column captures the possible emergent hazardous behavior from the interaction of the agents (e.g., that a collision could occur between spacecraft if a spacecraft does not change its position when other spacecrafts are expecting it to).

- iv. Give the criticality (e.g., critical, major, average, minor, etc.) of this failure as determined by the global effect of this failure on the system as a whole in the “Criticality” column.
- c. Apply any additional failure modes not captured by the provided keywords relevant to the current event and fill in the SFMECA row as appropriate.

The results of the application of the procedure detailed above to the *CollisionProtector* variation point’s events listed in its *Protocols and Activities* section of the Variation Point Schema are shown in Table 3 Table 4 and Table 5 for the *Analyze3DModel*, *DetectNearbySpacecraft* and *MoveToAvoidCollision* events.

For example, the SFMECA table for the *MoveToAvoidCollision* event of the *CollisionProtector* variation point of the Prospecting Asteroid Mission (PAM) case study, shown in Table 5, describes the local and system-wide effects of the “halt/abnormal termination”, “omission”, “incorrect logic/event” and “timing/order” failures of the *MoveToAvoidCollision* event. Each of these failures describe the effect on the local data and other events of the variation point and how those can propagate to the system level and potentially cause a collision between spacecraft in the PAM swarm. The system-wide effects for the failures of this event are classified at a criticality level of either Major or Critical and will likely require mitigation requirements to ensure that such failures are not possible in the MAS-PL, as discussed in Section 5.2.3.

Table 3 A Portion of the SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	Analyze3DModel	Halt/Abnormal Termination	The position and model of a nearby asteroid stored in the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data vector may be incomplete or partially incorrect. This may affect other events such as MonitorNearbyAsteroids and MoveToAvoidCollision.	The spacecraft's inaccurate mental model of the nearby asteroid could cause it to maneuver itself too close to the asteroid causing a collision.	Major
			Omission	The role fails to analyze the 3D model of a nearby asteroid potentially causing the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data to be incomplete or incorrect. This may affect other events such as MonitorNearbyAsteroids and MoveToAvoidCollision.	The failure to analyze the 3D model provided of a nearby asteroid(s) may cause the asteroid to incorrectly maneuver itself too close to an asteroid and cause a collision.	Critical
			Incorrect Logic/Event	The role incorrectly analyzes the 3D model of a nearby asteroid that may cause the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data to be incomplete or incorrect. This may affect other events such as MonitorNearbyAsteroids and MoveToAvoidCollision.	The spacecraft uses an inaccurate 3D model of a nearby asteroid that may cause it to maneuver itself into a nearby spacecraft or asteroid.	Critical
			Timing/Order	The role fails to analyze the 3D model of a nearby asteroid causing the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data to be outdated. The <i>riskForSystemFactor</i> data may be inaccurate since it was calculated based on outdated data. This may affect other events such as MonitorNearbyAsteroids, EvaluateRiskToGoal and MoveToAvoidCollision.	The spacecraft uses an outdated 3D model of a nearby asteroid(s) and may not be able to react in time to avoid a collision with an asteroid if the 3D model is not updated as expected.	Major

Table 4 A Portion of the SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	DetectNearby Spacecraft	Halt/Abnormal Termination	The role fails to complete its analysis of detecting nearby spacecraft and may not be aware of all nearby spacecraft. Thus, the data stored in <i>riskForSystemFactor</i> , <i>subswarmVector</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and <i>neabySpacecraft</i> may be inaccurate, corrupted or outdated.	The spacecraft does not have a full knowledge of all nearby spacecraft and may unknowingly maneuver itself into another spacecraft causing a collision. The spacecraft's ability to negotiate collision avoidance with another spacecraft using the NegotiateCollision Avoidance protocol can not be trusted by other spacecraft since the spacecraft's mental model of nearby spacecraft is not accurate.	Major
			Omission	The role fails to detect its surrounding for nearby spacecraft and may not be aware of all nearby spacecraft. The data stored in <i>riskForSystemFactor</i> , <i>subswarmVector</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and may be inaccurate or outdated and the <i>neabySpacecraft</i> may be incorrect or outdated.	The spacecraft has no knowledge of the positions of other nearby spacecrafts possibly causing the spacecraft to maneuver too close to another spacecraft causing a collision. The lack of knowledge of the positions of nearby spacecrafts may also cause the spacecraft's ability to avoid collisions using the Negotiate CollisionAvoidance protocol is using incomplete or inaccurate data.	Critical
			Incorrect Logic/Event	The role possible wrongly detects or miscalculates the positions of nearby spacecraft. The data stored in <i>riskForSystemFactor</i> , <i>subswarmVector</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and may be inaccurate or outdated and the <i>neabySpacecraft</i> may be incorrect or outdated.	The spacecraft's belief of the positions of other nearby spacecraft is inaccurate and it may collide into nearby spacecraft if maneuvers itself. The lack of knowledge of the positions of nearby spacecrafts may additionally cause the spacecraft's ability to avoid collisions using the Negotiate CollisionAvoidance protocol is using incomplete or inaccurate data.	Critical
			Timing/Order	The detection of nearby spacecrafts is delayed so that the role may not possible have the accurate locations of nearby spacecraft when it is expecting it. Because of this, the data stored in <i>riskForSystemFactor</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and may be inaccurate or outdated and the <i>neabySpacecraft</i> may be incorrect or outdated without the role knowing this.	The spacecraft may believe that the positions of nearby spacecraft it has stored in <i>subswarmVector</i> and <i>spacecraftPos</i> is correct and thus may inadvertently maneuver too close to another spacecraft and collide into it. The spacecraft may also provide inaccurate information to other spacecraft using the NegotiateCollision Avoidance protocol that may result in further collisions of spacecraft.	Major

Table 5 A Portion of the SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	MoveToAvoid Collision	Halt/Abnormal Termination	The <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data may be temporarily incorrect since the spacecraft did not complete moving to its new position. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft will not have moved to the position expected by other nearby spacecraft in the subswarm potentially causing a collision.	Major
			Omission	The spacecraft fails to move to its new assigned position in the subswarm possibly causing the <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data to be temporarily incorrect. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft will not have moved but, rather, maintain its previous position potentially causing a collision. Other spacecraft in the subswarm may expect the spacecraft to have moved to a new position which may cause a collision due to the discrepancies between actual and perceived spacecraft positions.	Critical
			Incorrect Logic/Event	The spacecraft fails to move to the position it is expecting possibly causing its <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data to be different than expected. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft moves to a position different than what it expects. Further, other spacecraft nearby will have expected the spacecraft to be in a different location potentially causing a collision.	Critical
			Timing/Order	The spacecraft fails to move to the new position until some later, undetermined time potentially causing its <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data to be different than expected. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft fails to move to the position it indicated to other spacecraft via the NegotiateCollisionAvoidance protocol at the time expected by the other spacecraft. This may cause a collision.	Major

5.2.2 Constructing a SFMECA Data Table for a Variation Point in Gaia-PL

Constructing the SCMECA data table for a variation point in Gaia-PL is identical to that of constructing a SFMECA event table except for the failure mode keywords used. For completeness, the procedure to construct a SFMECA table for the data from the Variation Point Schema(s) using the event guidewords consists of the following steps:

1. For each role:
 - a. Create a new SFMECA data table similar to that shown in Table 6 and fill in the role's name and its possible variation points.
 - b. Then, for each of the variation points possible, listed in the role's Role Variation Points Schema (see Section 4.2.2.2):
 - i. For each of the pieces of data listed in the *Permissions* section of the Variation Point Schema:
 - a. Provide the event name in the "Data" column.
 - b. Apply each of the failure mode keywords (i.e., "incorrect value", "absent value", "wrong timing" and "duplicated value") to the data. For each keyword:
 - i. Provide the data failure mode keyword in the "Failure Mode" column.
 - ii. Describe the possible local effect(s) if the keyword failure happened to the event under consideration in the "Local Effect(s)" column. The local effect will likely only affect this role or this agent and its description should not include the propagation of its failure to other agents or components of the global system.

- iii. Describe the possible system-level effect(s) if the keyword failure mode occurred in the “System Effect(s)” column. This column captures the possible emergent hazardous behavior from the interaction of the agents (e.g., that a collision could occur between spacecraft if a spacecraft does not change its position when other spacecrafts are expecting it to).
 - iv. Give the criticality (e.g., critical, major, average, minor, etc.) of this failure as determined by the global effect of this failure on the system as a whole in the “Criticality” column.
- c. Apply any additional failure modes not captured by the provided keywords relevant to the current data and fill in the SFMECA row as appropriate.

The results of the application of the procedure detailed above to the *CollisionProtector* variation point’s data listed in its *Permissions* section are shown in Table 6, Table 7 and Table 8 for the *nearbyAsteroids*, *nearbySpacecraft* and *position* data.

For example, the SFMECA table for the *position* data, shown in Table 8, describes the local and system-wide effects of the “incorrect value”, “absent value”, “wrong timing” and “duplicated value” failures of the *position* data. Each of these failures describe the effect on the local events and other data of the variation point and how those can propagate to the system level and potentially cause a collision between spacecraft and asteroids. The system-wide effects for the failures are classified at a criticality level of either Major or Critical and will likely require mitigation requirements to ensure that such failures are not possible in the MAS-PL, as discussed next.

Table 6 A Portion of the SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	nearbyAsteroids	Incorrect Value	The variation point belief of the positions of nearby asteroids may be incorrect. The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect and the Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the wrong data. The <u>RequestAsteroidPositions</u> protocol may provide inaccurate information upon request.	The spacecraft will use incorrect values of the locations of nearby asteroids and may unknowingly maneuver too close to an asteroid and collide with it. The spacecraft may also provide the incorrect information to other spacecraft through the <u>RequestAsteroidPositions</u> protocol causing other spacecraft to potentially collide into an asteroid. The incorrect data may also invalidate the scientific data collected on the asteroids.	Critical
			Absent Value	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or corrupted since no location values for nearby asteroids were available. The Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the unavailable data.	The spacecraft will have no information on the location of nearby asteroids and will need to request the locations via the <u>RequestAsteroidPositions</u> protocol. May cause a collision with an asteroid since the locations are unknown. May corrupt some of the scientific data collected on the asteroids or cause the execution of the variation point to freeze.	Major
			Wrong Timing	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or outdated since the location of nearby asteroid data is old. The Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may result in outdated output.	The spacecraft may have made maneuvering decisions based on outdated information of the location of nearby asteroids. This may cause a collision with an asteroid since the locations are outdated.	Major
			Duplicated Value	The Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may be uneedingly executed twice since the data was updated twice.	The spacecraft will may have had to execute the Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events twice possibly delaying the response to request from other spacecraft.	Minor

Table 7 A Portion of the SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	nearbySpacecraft	Incorrect Value	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or corrupted since no location values for other nearby spacecraft are available. The DetectNearbySpacecraft, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the incorrect.	The spacecraft will use incorrect values of the locations of nearby spacecraft and may unknowingly maneuver too close to another spacecraft and collide with it. The spacecraft may also provide the incorrect information to other spacecraft through the <u>RequestSpacecraftLocations</u> protocol causing other spacecraft to potentially collide.	Major
			Absent Value	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be missing or corrupted since no location values for other nearby spacecraft are available. The DetectNearbySpacecraft, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the unavailable data.	The spacecraft will have no information on the location of nearby spacecraft and will need to request the locations via the <u>RequestSpacecraftLocations</u> protocol. May cause a collision with an spacecraft since the locations are unknown.	Critical
			Wrong Timing	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or outdated since the location of nearby asteroid data is old. The DetectNearbySpacecraft, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the outdated data.	The spacecraft may have made maneuvering decisions based on outdated information of the location of nearby spacecraft. This may cause a collision since the locations are outdated.	Critical
			Duplicated Value	The EvaluateRiskToGoal and MoveToAvoidCollision events may be unneedingly executed twice since the data was updated twice.	The spacecraft may report to others that it is malfunctioning since it received duplicated values.	Minor

Table 8 A Portion of the SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	position	Incorrect Value	The variation point uses the incorrect value of its current position possibly affecting the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearby Asteroids and MoveToAvoidCollision events. This may also cause the variation point to incorrectly change its <i>riskForSystemFactor</i> data and generate inaccurate <i>collisionRiskFactor</i> , <i>riskToGoal Factor</i> , <i>nearby Asteroids</i> and <i>nearbySpacecraft</i> data.	The spacecraft does not know its actual position and may report a false position to other spacecraft via the <u>RequestSpacecraftLocations</u> protocol potentially causing a collision.	Critical
			Absent Value	The missing or corrupted value of its current position may affect the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids and MoveToAvoidCollision events since the data is unusable. This may also cause the variation point to corrupted its <i>riskForSystemFactor</i> data and generate corrupted <i>collisionRiskFactor</i> , <i>riskToGoalFactor</i> , <i>nearbyAsteroids</i> and <i>nearbySpacecraft</i> data.	The spacecraft does not know its actual position and may report a false position to other spacecraft via the <u>RequestSpacecraftLocations</u> protocol potentially causing a collision. Alternatively, the spacecraft uses the missing or corrupted value and may collide into a nearby spacecraft.	Critical
			Wrong Timing	The variation point uses the outdated value of its current position possibly affecting the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids and MoveToAvoidCollision events. This may also cause the variation point to incorrectly change its <i>riskForSystem Factor</i> data and generate outdated <i>collisionRiskFactor</i> , <i>riskToGoal Factor</i> , <i>nearbyAsteroids</i> and <i>nearbySpacecraft</i> data.	The spacecraft may have made maneuvering decisions based on outdated information of position potentially causing a collision.	Major
			Duplicated Value	The variation point uses the duplicate position information to execute the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids and MoveToAvoidCollision events twice.	The spacecraft may report to others that it is malfunctioning since it received duplicated values of its current position.	Minor

5.2.3 Deriving Safety Requirements from the SFMECA Tables

The development of the Software Failure Modes, Effects and Criticality Analysis (SFMECA) event and data tables for the variation points of a role identifies the possible local and system effects of the failure of an event. In addition, the creation of the SFMECA tables provides the software engineer with the opportunity to identify missing safety requirements, discover the interactions and relationships between the events and data and/or verify the system design and requirements specifications.

A SFMECA starts with the failure of a software component or subsystem, detailed in the “Local Effects” column, and then looks at its effect on the overall system, documented in the “System Effects” column. Applying the structured procedure to create the SFMECA for a multi-agent system product line (MAS-PL) in the Gaia-PL methodology, as described in Sections 5.2.1 and 5.2.2, yields a list of possible accidents that could compromise the success of the system along with their potential causes. This analysis may reveal safety requirements that should be added to the variation points of a role to prevent the propagation of the failure to the system level.

For example, the “omission” failure mode keyword of the *MoveToAvoidCollision* event of the *CollisionProtector* variation point, shown in Table 5, revealed in this case study that some verification was needed for the spacecraft itself and the nearby spacecraft that it had indeed maneuvered to the desired position. To achieve this, a *DetermineNewPosition* event and a *RequestVerifyPosition* protocol could be added to provide the variation point with this needed functionality. This additional functionality, not included in the original requirements, could better prevent the spacecraft and nearby spacecraft from incorrectly assuming the location of a spacecraft and may avoid collisions.

Similarly, the “wrong timing” failure mode keyword of the *nearbySpacecraft* data of the *CollisionProtector* variation point, shown in Table 7, illustrated the need for timestamps to be included with the position data of nearby spacecraft so that each spacecraft can assess the freshness of the data. This information, not originally included in the requirements, may allow the *CollisionProtector* variation point to better prevent collisions with nearby spacecraft.

The application of the SFMECA to the requirements specifications of a MAS-PL in Gaia-PL provides insight into missing requirements that may be needed to prevent the propagation of the failure from the local level to a system-wide level. It was found that the missing safety requirements discovered in this process were often not considered during the development of the requirements specifications of the MAS-PL from its requirements. Further, the structured process described to derive the SFMECA for a safety-critical MAS-PL.

The application of the failure mode keywords to each event and data of a role variation point requires deep consideration of their possible interaction and effect on the other data and events. This differs from the development of the original requirements specifications, described in Section 4.2.2, from the MAS-PL requirements documented in the Commonality and Variability Analysis (CVA), described in Section 4.2.1, since a better understanding of the relationships between events and other events as well as events and data is needed.

Finally, the creation of a SFMECA provides some verification of the design of the variation points and their requirements specifications in that it further reveals the interactions of the events and data of a variation point and requires the developer to better think about the variation point’s functionality and its effect on the entire system. Further, it allows a software engineer to identify the hazardous states that the MAS-PL may enter

and provide them with the chance to derive requirements to prevent a system failure or verify that the existing mitigation requirements will avoid the failure. This provides assurances that certain classes of failure modes that might occur in individual agents will not produce unacceptable effects in the composite system and demonstrates the ability of a variation points failure-monitoring and failure mitigation software tasked with the system safety requirements to safety standards.

The opportunity to derive further safety requirements and better demonstrate the compliance of the design in handling hazardous situations for a MAS-PL using the SFMECA generated in this section will be discussed in Section 5.4 as a part of the Bi-Directional Safety Analysis (BDSA) technique.

5.2.4 Deriving the SFMECA Tables for a Specific Product in a MAS-PL

Sections 5.2.1 and 5.2.2 described the creation of the SFMECA event and data tables for the variation point of a role in Gaia-PL. The creation of the SFMECA safety analysis asset occurs in Gaia-PL's Analysis and Design Phase, as shown in Figure 30, and uses Gaia-PL's Variation Point Schemas. The Analysis and Design Phase of Gaia-PL occurs within the domain engineering phase of Weiss and Lai's Family-Oriented Abstraction, Specification, and Translation (FAST) model [88]. Thus, the SFMECA derived represents all the roles and variation points possible in any agent of the multi-agent system product line (MAS-PL).

In Gaia-PL's Detailed Design Phase (FAST's application engineering phase), an agent is designed and developed by selecting the roles and each role's set of possible variation points for a specific agent, as described in Section 4.2.3. The SFMECA tables produced by following the structured procedure of Sections 5.2.1 and 5.2.2 will produce SFMECA tables not relevant to a specific agent (e.g., the agent does not contain a role or

variation point documented in the SFMECA). Thus, any given agent's SFMECA safety analysis artifacts will be a subset of the SFMECA tables.

The partitioning of the SFMECA tables by the roles and variation points, described in Sections 5.2.1 and 5.2.2 eases the derivation of the created SFMECA tables during the design and development of specific agent through reuse. The SFMECA for a specific agent can be derived by simply including the roles and variation points that are possible in the agent and discluding the roles and variation points not possible in the agent. For example, in the Prospecting Asteroid Mission (PAM) case study used in this dissertation, any agent with the *CollisionProtector* variation point would include the SFMECA tables given in Table 3, Table 4, Table 5, Table 6, Table 7 and Table 8. Similarly, any PAM agent not including the *CollisionProtector* variation point would not have any of these tables.

This process can provide the SFMECA safety analysis assets for all allowable configurations of an agent in a MAS-PL (e.g., all 160 possible spacecraft in the PAM case study).

5.2.5 Accommodating MAS-PL Evolution in the SFMECA in Gaia-PL

In Section 4.3.2, we discussed the evolution of a multi-agent system product line (MAS-PL) in Gaia-PL. A MAS-PL can evolve in three ways relevant to this work: 1. new agents may be added to the system; 2. new roles with new functionality may be created that future agents can employ; and 3. new variation points may be added to existing roles that future agents can employ. To consider the safety consequences of the new functionality of the evolved MAS-PL, the Software Failure Modes, Effects and Criticality Analysis (SFMECA) must be updated.

The addition of a new agent(s) with no new functionality into an already deployed MAS-PL only necessitates the inclusion of the SFMECA event and data tables for the

roles and the variation points that are included in the agent. The partitioning of the SFMECA tables by the roles and variation points, described in Sections 5.2.1 and 5.2.2, allows the specific agent's safety analysis artifacts to be described by simply selecting the relevant SFMECA tables, as was discussed in Section 5.2.4.

The inclusion of a new role(s) into a MAS-PL requires the addition of new functionality not included in the original deployment of the agents in the MAS-PL. The new roles added to the MAS-PL can then be included in future agents of the system. The failure to assess the new agents for potential hazards may compromise the entire MAS-PL. Thus, the inclusion of a new role necessitates the SFMECA safety analysis to be updated to include the functionality of the new role and the possible system-wide effects of the functionality of the new role if it fails. The inclusion of a new role into a MAS-PL will often require the inclusion of new variation points to implement the functionality of the new role.

To accommodate the inclusion of new role(s) and/or variation points into a MAS-PL, the SFMECA must be updated to reflect the updates. First, the new Gaia-PL Role Variation Points Schema(s) and Variation Point Schema(s) must be developed to document the new requirements specification for the new role(s) and/or variation points, as described in 4.3.2. After documenting the new functionality, the SFMECA event and data tables for the new variation points can be derived using the structured process described in Section 5.2.1 (to create the SFMECA event tables) and Section 5.2.2 (to create the SFMECA data tables).

These steps will accommodate the types of evolution possible in a MAS-PL so that the SFMECA safety asset can be updated and used for future versions of agents of the system.

5.2.5 Discussion

The structured process to derive and document the Software Failure Modes, Effects and Criticality Analysis (SFMECA) of a multi-agent system product line (MAS-PL) from Gaia-PL's Variation Point Schemas inherits the reusability of the Gaia-PL methodology and can accommodate all allowable configurations of an agent in a MAS-PL (e.g., all 160 possible spacecraft in the PAM case study).

The partitioning of the SFMECA tables by the roles and variation points of a MAS-PL similarly associates each variation point with a set of SFMECA event and data tables. For those roles and variation points that are common in every agent (e.g., the *Navigator* role, discussed in Section 4.2.2.1), the SFMECA tables will always be included as a part of the safety analysis assets of an agent. However, for those roles and variation points that may or may not be included in an agent (e.g., the "Leader" variation point of the *Self-Optimizer* role, discussed in Section 4.2.2.2), the SFMECA tables will not always be included as a part of the safety analysis assets of an agent. Thus, the SFMECA safety analysis assets created using the approach described in this section are reusable for the agents of a MAS-PL in the same way that the Variation Point Schemas, discussed in Section 4.2, are reusable.

The structured process to derive and document the SFMECA of a MAS-PL from Gaia-PL's Variation Point Schemas could be applied, without change, to the Role Schema used in Gaia. Yet, the inability of Gaia to hierarchically capture the variation points of a role, as described in Sections 4.1.2.3, and its inability to partition the common and variable portion of role, as is done in Gaia-PL's variation points, the SFMECA tables would create a large amount of redundancy to capture the failure modes and effects of the redundantly documented functionality, as described in Section 4.4.2. Thus, the reusability and development cost would be lessened, similarly to that described in Section 4.4.2,

using Gaia-PL and the SFMECA process described in this section compared to if it was applied to the Role Schema's in the Gaia methodology.

5.2.6 Summary

This section discussed our adaptation of Software Failure Modes, Effects and Criticality Analysis (SFMECA) in our Gaia-PL Agent-Oriented Software Engineering (AOSE) methodology to produce a safety analysis technique specifically for safety-critical MAS-PL. We provided a structured process to analyze the Variation Point Schemas produced in the Gaia-PL methodology to discover the ways in which events and data of an agent can fail and the effects of the failures on the entire system.

The SFMECA captures the propagation of undesirable behavior in the MAS-PL. That is, the SFMECA process described in this section describes a failure at a local level (i.e., the role or variation point of a single agent) and details the possible consequences of the propagation of this failure at a system-wide level (i.e., the collection of agents in a multi-agent system). It is important to capture such behavior in a MAS-PL so that the collected behavior of the system is known and precautions can be made to prevent undesirable behavior.

The SFMECA can also aid in discovering missing safety requirements, designing mitigation requirements to prevent failures and verify existing safety requirements. Finally, we illustrated how the SFMECA safety asset can be reused for a specific agent given its roles and variation point and how the SFMECA can accommodate the evolution of a MAS-PL.

The next section describes the backward search, safety analysis technique used in this work, Product-Line Software Fault Tree Analysis (PL-SFTA) and its tool PLFaultCAT.

5.3 Product-Line Software Fault Tree Analysis and PLFaultCAT

Section 5.2 discussed our forward-based, safety analysis technique, an adapted Software Failure Modes, Effects and Criticality Analysis (SFMECA), for a safety-critical multi-agent system product line (MAS-PL). The SFMECA starts with the failure of a role's variation point and then looks at its effect on the overall system. However, the results of a forward analysis, such as SFMECA, may not cover all possible hazards of a system and fail to consider the combination of multiple events and their effect on possible system-wide hazards [44]. For a safety-critical system, it is also often necessary to perform a backward-based, safety analysis to better ensure that hazardous states and their causal events are identified and mitigated against.

This section details the backward analysis search technique, product-line software fault tree analysis (PL-SFTA), and its tool, PLFaultCAT (**P**roduct-**L**ine **F**ault Tree **C**reation and **A**nalysis **T**ool), that we have developed and used in this work to analyze a safety-critical MAS-PL. This section offers additional assurance to software engineers designing and developing a safety-critical MAS-PL by providing a tool-supported software safety analysis technique. PLFaultCAT is an interactive, partially-automated software support application to aid software engineers with the visualization and pruning process of a PL-SFTA. Specifically, the tool exploits the reusability inherent in product-line engineering by deriving reusable safety analysis assets (i.e., the product-line members' fault trees) for future systems within the existing product line.

5.3.1 Product-Line Software Fault Tree Analysis Overview

The product-line Software Fault Tree Analysis (PL-SFTA) maintains the safety analysis qualities of traditional Software Fault Tree Analysis (SFTA) while accommodating reusability in product-line engineering. Traditional SFTA targets the safety analysis of potentially harmful states for a single product. The PL-SFTA, however,

incorporates the variabilities among the different members of a product line and contributes to the safety analysis for the entire product line without performing traditional SFTA serially on each product-line member. A new SFTA for a product line member can be derived almost automatically with PLFaultCAT using its pruning algorithm. The aim of this technique and tool is to support the confident reduction of the safety analysis needed on a new product in the product line and, ultimately, a less expensive and shorter product development process.

Section 5.3 illustrates how and to what extent the PL-SFTA technique, supported by the PLFaultCAT tool, can be used by software engineers as a reusable safety analysis for designing and developing a multi-agent system product line (MAS-PL). Like the Gaia-PL methodology detailed in Chapter 4 and the SFMECA approach described in Section 5.2, the PL-SFTA technique employs the Family-Oriented Abstraction, Specification, and Translation (FAST) model's domain and application engineering phases [88]. In the domain engineering phase, the PL-SFTA is constructed with the aid of the PLFaultCAT tool. The application engineering phase develops and performs the safety analysis on new product-line members (i.e., the agents of a MAS-PL). The construction of a PL-SFTA, aided by PLFaultCAT, during the domain engineering phase provides the means for reusing the PL-SFTA for new members (i.e., agents of a MAS-PL). Within the application engineering phase we utilize PLFaultCAT to facilitate the derivation of new product-line members' fault tree(s).

Figure 32 provides an overview of the construction and derivation process of a PL-SFTA within the two-phased FAST approach. The role of PLFaultCAT in this framework primarily resides in the application engineering phase. Although PLFaultCAT can assist in the initial graphical representation of a product-line fault tree, the chief contribution of the PLFaultCAT tool is to automatically produce the fault tree artifacts that software engineers desire at the end of the application engineering phase.

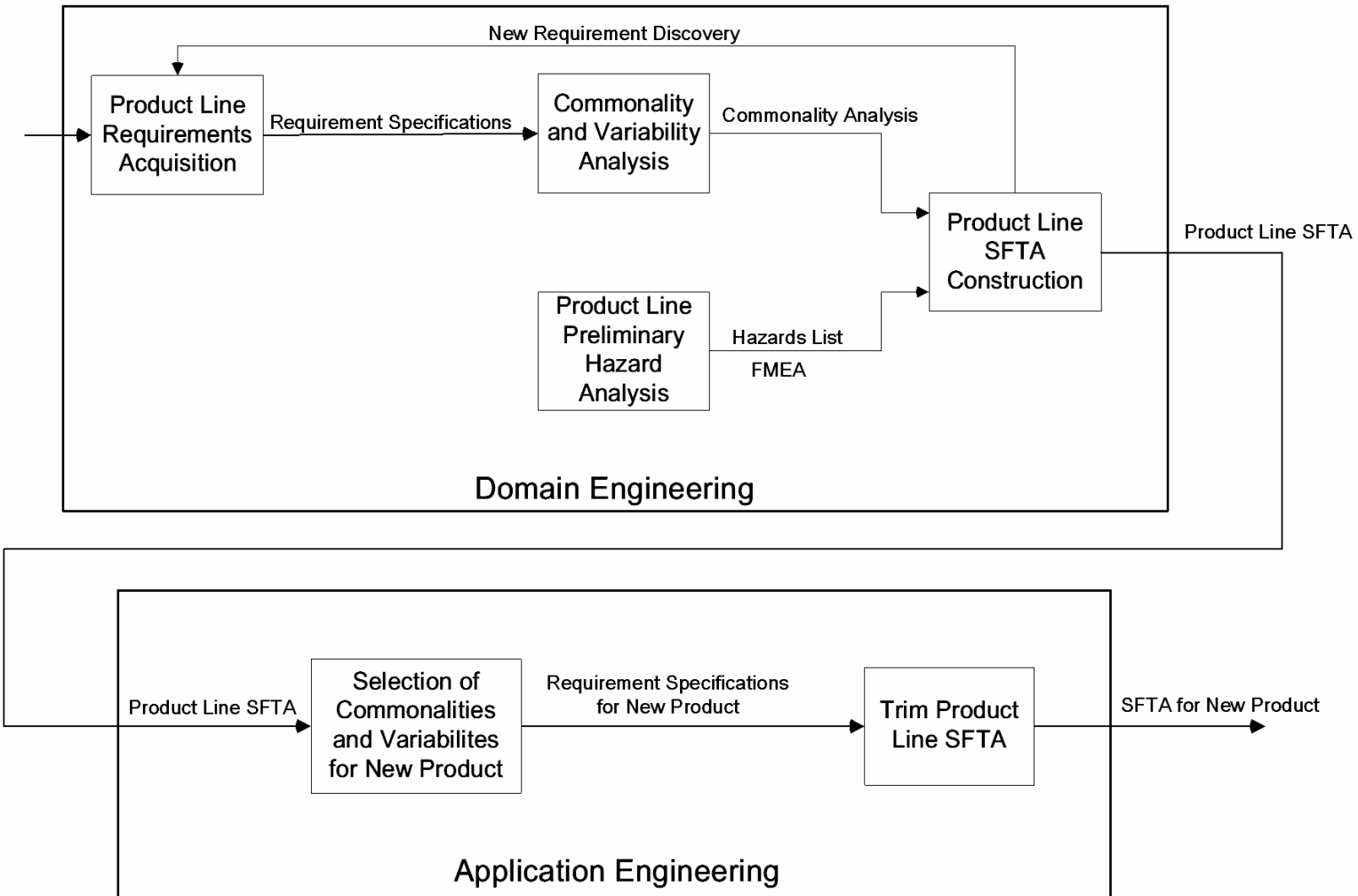


Figure 32 An Overview of the PL-SFTA Safety Analysis Technique

To assist in the creation of a PL-SFTA, PLFaultCAT can utilize DECIMAL [23], [58], [59] (described in Section 2.1.4 and used within the Gaia-PL methodology in Section 4.2.1.2) to aid in:

- Documenting a MAS-PL's commonalities, variabilities and dependencies
- Defining an agent of a MAS-PL through the selection of variabilities
- Automatically verifying consistency of a new agent with the MAS-PL's dependencies

PLFaultCAT can then link to the requirements defined in DECIMAL to associate with a PL-SFTA's leaf nodes. The use of DECIMAL in conjunction with PLFaultCAT provides better management, traceability and automated verification of a product-line's requirements as well as the creation, derivation and analysis of a PL-SFTA.

In addition to aiding the creation of a PL-SFTA and the derivation of the SFTA for a product-line member, PLFaultCAT provides several automated safety analyses to identify failure points and safety-critical requirements. Figure 33 provides an overview of these automated safety analysis as well as the overview of DECIMAL and PLFaultCAT's role in the design and development of a safety-critical product line. A *minimum-cut set analysis* analyzes a single PL-SFTA and identifies the smallest sets of events that must occur such that the root node accident will occur [44]. A *probability report* calculates the probability of occurrence of the root node given the probabilities of all other nodes. A *single-point failure analysis* searches the set of SFTAs for single-point failures (i.e., those hazards connected by a logical OR gate in the SFTA) at user-specified depth [44]. A *variability failure contribution analysis* analyzes all the PL-SFTAs to find those variabilities or combination of variabilities that contribute to a high number of hazards.

The single-point failure analysis and variability failure contribution analyses, in particular, can aid in identifying latent safety requirements. For example, a single-point failure found in a product-line SFTA may necessitate new safety requirements (e.g., a

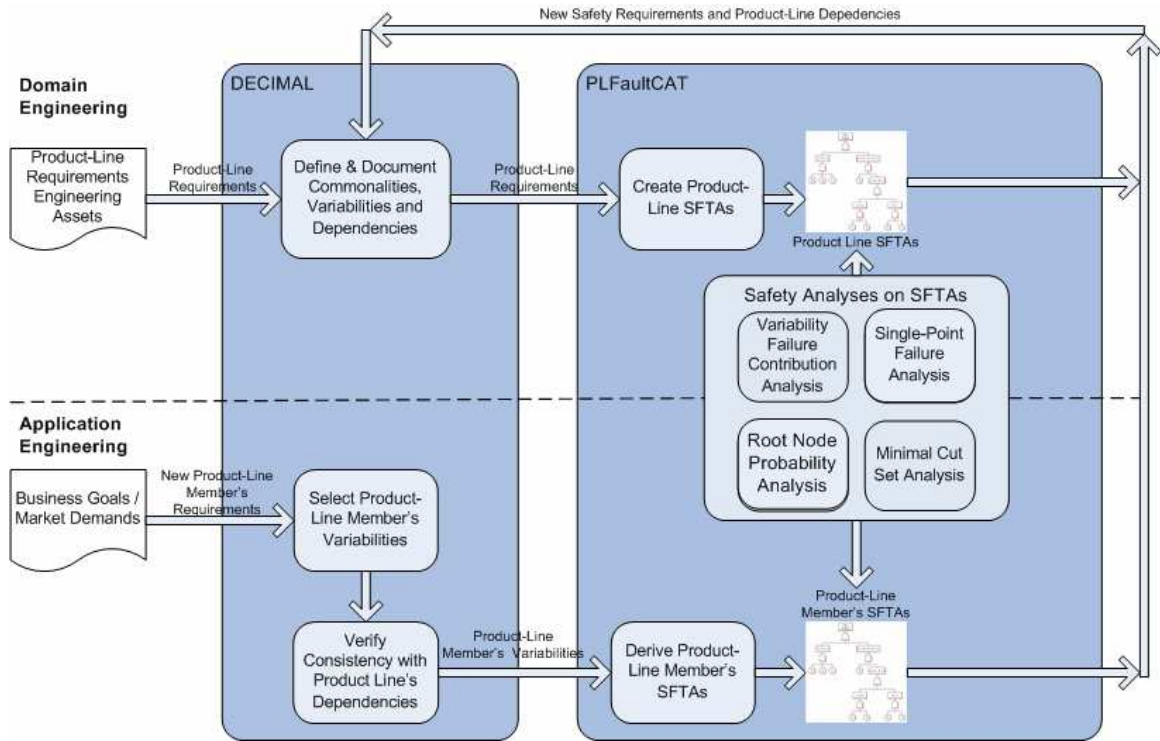


Figure 33 An Overview of DECIMAL and PLFaultCAT's Role in the Design and Development of Safety-Critical Product Lines

safety guard) to transform the single-point failure (i.e., an OR gate) into a non single-point failure (i.e., an AND gate). Similarly, the variability failure contribution analysis may indicate variabilities that should not be allowed to be present in any product-line member (i.e., a product-line dependency).

The remainder of Section 5.3 details PLFaultCAT's software architecture, the creation of a PL-SFTA for a product line, the derivation of a SFTA for a product-line member and the additional safety analysis opportunities available using PLFaultCAT illustrated using the Prospecting Asteroid Mission (PAM) case study.

Note that although the work described in this section illustrates our PL-SFTA technique and its tool, PLFaultCAT, on a MAS-PL, we describe its application to a general product line. Section 5.3.6 delineates a specific, alternative approach using PL-SFTA and PLFaultCAT to perform the safety analysis of a MAS-PL. Except for Section 5.3.6, the work described in Section 5.3 is applicable to any safety-critical software product line. Our papers in [17], [18], [24], [45], [47], [48] partially illustrate the PL-SFTA technique using a traditional product line application (i.e., not a MAS-PL).

5.3.2 PLFaultCAT Overview and Software Architecture

This section introduces and briefly describes the PLFaultCAT tool. PLFaultCAT is the software tool developed to aid in both the domain engineering phase for initial product-line software fault tree analysis (PL-SFTA) development and representation as well as in the application engineering phase for the derivation of product line members' software fault tree(s) from the PL-SFTA developed in the domain engineering phase. In this section, we present an overview of the PLFaultCAT tool and give a description of its software architecture.

5.3.2.1 PLFaultCAT Overview

PLFaultCAT (**P**roduct-**L**ine **F**ault Tree **C**reation and **A**nalysis **T**ool) is a tool-assisted visualization and pruning application for the creation and analysis of product-line software fault trees. PLFaultCAT is an extension of the FaultCAT application [4]. FaultCAT is an open-source fault tree creation tool written in Java that is primarily geared towards analyzing a system for faults to determine how faults can affect other parts of the system [4]. FaultCAT does this by attaching fault probabilities to each node. FaultCAT provides a user the ability to graphically construct and represent the nodes and logic gates of a traditional fault tree. A complete discussion of the construction of a PL-SFTA using PLFaultCAT is given in Section 5.3.3.

PLFaultCAT internally stores the fault trees in an XML format, making it easy to manipulate and alter. This is important because product lines routinely evolve, and the safety analysis must accordingly be updated. PLFaultCAT builds on the existing XML storage format of a fault tree in FaultCAT. PLFaultCAT utilizes the XML DOM parser to perform the pruning necessary to generate a product-line member's fault tree(s) from the PL-SFTA during the pruning process of the application engineering phase. A full discussion of the pruning algorithm and how it is handled in PLFaultCAT is given in Section 5.3.5. In addition to the graphical and XML view of the fault tree, PLFaultCAT presents a textual overview of a fault tree that lists the nodes of a fault tree, the type of a leaf node (either a commonality or variability) and the value of a leaf node commonality or variability.

5.3.2.2 PLFaultCAT Software Architecture

The PLFaultCAT software architecture is built directly upon the software architecture of the original FaultCAT application. Thus, the majority of the PLFaultCAT tool inherits the base software architecture of FaultCAT. PLFaultCAT enhances FaultCAT by adding onto the software architecture the functionality needed to accommodate the creation and analysis of a PL-SFTA. Figure 34 shows the architecture of the PLFaultCAT application.

PLFaultCAT maintains all the functionality of FaultCAT and can still accommodate the creation and analysis of a single product software fault tree. To achieve this, the original FaultCAT software architecture, including the class structures, is maintained. Any additional functionality added to the already existing classes of FaultCAT has been tested to ensure that it does not interfere with FaultCAT's intended functionalities.

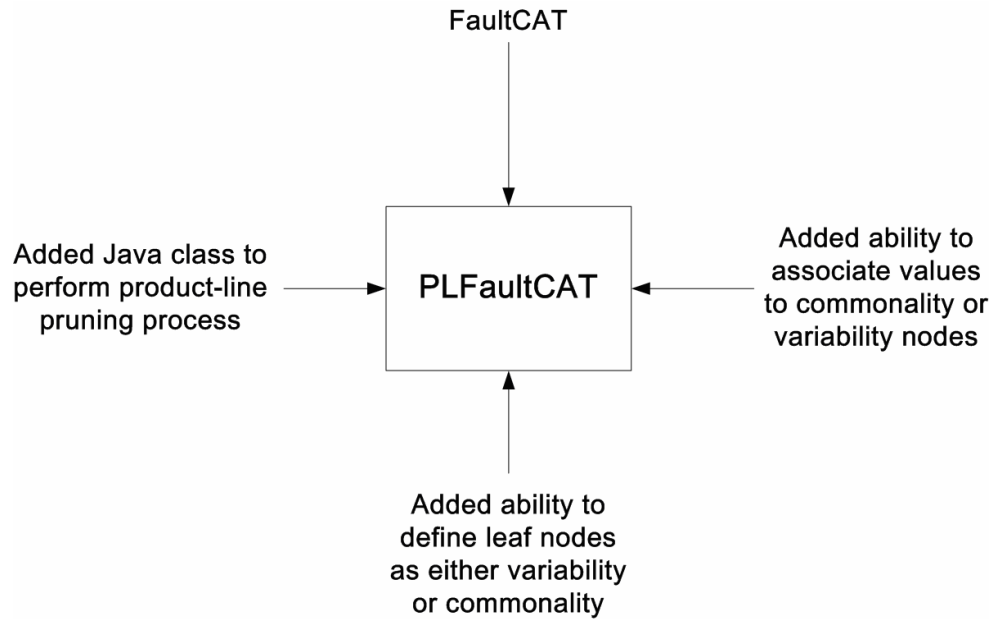


Figure 34 PLFaultCAT's Software Architecture

5.3.2.3 Implementation of PLFaultCAT

The major contribution to the PLFaultCAT tool is to add the nearly automatic pruning process of deriving a product-line member's fault tree from the PL-SFTA. Within PLFaultCAT, this was implemented as additional Java classes not found in FaultCAT. These Java classes provide the interactive, GUI-driven interface to allow a user to actively select the variabilities to include in any new product-line member. The selected variabilities then are used to properly prune the stored PL-SFTA to produce the derived product-line member's software fault tree.

To facilitate the creation of a PL-SFTA, PLFaultCAT provides the ability to define a leaf node within a fault tree to be a fault associated with either a commonality requirement/component or a variability requirement/component. Defining leaf nodes as either being coupled to a commonality or a variability allows for the pruning process to determine which branches or subtrees are relevant for a given fault tree and a selected set of variabilities.

PLFaultCAT provides the ability to specify the value(s) for a particular commonality or variability comprising the product line. Assigning the value(s) of a particular commonality or variability to a leaf node within a fault tree provides (1) an association of the leaf node with specifically what the choice of variability must be in order to contribute to its parent event node and its associated subtree and (2) a heuristic for the pruning algorithm to resolve those branches or subtrees that are applicable for a given fault tree and a selected set of variabilities and their values.

Lastly, PLFaultCAT was originally developed as a tool separate from DECIMAL (described in Section 2.1.4 and used within the Gaia-PL methodology in Section 4.2.1.2) as described in [24]. PLFaultCAT and DECIMAL have now been integrated and extended in order to provide software engineers a single solution to requirements management and automated software safety analyses across the product-line lifecycle. The new features included in PLFaultCAT in this integration include:

- Linking product-line requirements and verified product-line members from DECIMAL to fault tree nodes in PLFaultCAT
- Automating the derivation of the SFTAs of a new product-line member from the set of product-line SFTAs in PLFaultCAT
- Automating a user-defined, single-point failure analysis for the set of product-line or product-line member SFTAs
- Automating the analysis and identification of the variability failure contribution analysis to identify safety-critical requirements for the product line

These additional features required further functionality, implemented in Java, to be included in PLFaultCAT that was not originally found in the FaultCAT tool.

5.3.3 Constructing a Product-Line Software Fault Tree

This section details the construction of a product-line Software Fault Tree Analysis (PL-SFTA) for a safety-critical software product line. The creation of the PL-SFTA for a software product line occurs during the domain engineering phase of the Family-Oriented Abstraction, Specification, and Translation (FAST) model [88]. In the Gaia-PL methodology, shown in Figure 30, the creation of a PL-SFTA for a multi-agent system product line (MAS-PL) occurs during the Analysis and Design Phase.

In this section, we first discuss the product-line requirements and the list of possible hazards needed to develop a PL-SFTA. We then provide the steps to construct a product-line software fault tree in PLFaultCAT.

5.3.3.1 Identifying Hazards for a PL-SFTA

As shown in Figure 32, the safety analysis for the domain engineering phase of product-line development typically results from a Preliminary Hazard Analysis (PHA). A PHA identifies the systems' hazards at an early stage of development with the aim of determining their impact on the system [44]. A domain hazards list will often exist prior to the development of the product line from historical data or domain expertise. If no preexisting hazards list is available, procedures exist to establish a workable, comprehensive list [29]. The creation of the hazards list requires extensive domain expertise and may be performed in parallel with the documenting of the software product-line requirements in a Commonality and Variability Analysis (CVA), described in Section 2.1.1 and detailed in the context of Gaia-PL in Section 4.2.1.1.

Alternatively, states from the "System Effects" column of a Software Failure Modes, Effects and Criticality Analysis (SFMECA), described in Section 2.3.2.2 and adopted for the use in Gaia-PL in Section 5.2, can be used as a source of hazards for the root nodes of the product-line Software Fault Tree Analysis (SFTA) as they represent

states that must be avoided. For example, for the Prospecting Asteroid Mission (PAM) case study used throughout this dissertation, possible hazards from the SFMECA table shown in Table 8 could be “A spacecraft to spacecraft collision occurred” and “A spacecraft to asteroid collision occurred”.

Following the initial product line requirements acquisition in the FAST method, a precise definition of a product line is achieved through the creation of a CVA, as described in Section 2.1.1 and detailed in the context of Gaia-PL in Section 4.2.1.1. Figure 5 and Figure 6 provide a portion of the CVA for the PAM case study that was used in Chapter 4 to develop the requirements specifications in the Gaia-PL methodology and will be used as a running example to illustrate the activities involved in the domain and application engineering phase use of PLFaultCAT and the PL-SFTA technique. In particular, Figure 5 and Figure 6 display the commonalities and variabilities associated with the safety-critical *SolarStormWarner* (discussed in Section 5.2.1 and shown in Appendix D, page 296) and *CollisionProtector* (discussed in Section 4.2.2.3 and shown in Appendix D, page 301) roles and their variation points. Table 2 gives a portion of the Parameters of Variation document detailing the allowable options for the variabilities listed in Figure 6.

A SFMECA, described in Section 5.2, searches the failure modes possible in the product line, determines their potential local effects and establishes their potential effects on the other members of the system [53]. Excerpts of the SFMECA for the PAM case study were given in Section 5.2.1. This portion of the SFMECA includes only those failure modes relevant to the possible collision of a multiple spacecraft or the collision of a spacecraft with an asteroid. Note that while this particular SFMECA concentrates mainly on the software failures of the PAM case study, it may also include those hardware failures (which will typically contribute as leaf nodes) that contribute to the propagation of software failures.

If a SFMECA exists for a product line, this analysis can produce the necessary domain knowledge to begin construction of the PL-SFTA using the prescribed steps detailed in the following section. If a SFMECA does not exist, construction of the PL-SFTA proceeds directly to Step 2 of Section 5.3.3.2 after assembling an intermediate node tree without the aid of a SFMECA. The following section describes our steps to construct a PL-SFTA for a safety-critical product line using the PAM MAS-PL case study.

5.3.3.2 Constructing a PL-SFTA for a Hazard

The construction of the product-line SFTA using PLFaultCAT proceeds through three basic steps:

Step 1. Determine the root node and generate the intermediate node tree. As explained previously and shown in Figure 32, the root node hazard of any SFTA often derives from a preexisting hazards list or a list generated during the Preliminary Hazard Analysis (PHA) phase, possibly from a Software Failure Modes, Effects and Criticality Analysis (SFMECA).

Causal events can be viewed as contributing events to the root node and are derived from the SFMECA or equivalent domain expertise. The SFMECA provides the causal events in the "Cause of Failure" or "Local Effect(s)" column as well as the potential contributing nodes leading to the causal event. (Note that some work, including [17], [24] and [44], has used a "Cause of Failure" column in place of a "Local Effect(s)" column to describe the origin of the failure mode. In this work, and our previous work in [22], we use the "Local Effect(s)" column to better indicate that the failure is originating from a role or variation point of an agent. However, the information contained in these columns is essentially identical.) Gathering the causal events, we construct an intermediate node tree to establish the cause-event hierarchy. The intermediate node tree,

while not necessary in the construction of a PL-SFTA, aids in jump-starting the organization and analysis of the PL-SFTA. Essentially, the intermediate node tree represents a typical fault tree without the Boolean logic gate relationships between causal events and effects. To determine the intermediate node tree using this process, we use the PL-SFTA_CREATE algorithm, shown in Figure 35, starting with the root node event as the initial *event*.

PL-SFTA_CREATE(*event*):

STEP 1 Create node in tree for *event*

STEP 2 If node is not root node then

STEP 2.1 Attach node to parent node

STEP 3 Scan SFMECA "Possible Effect(s)" column for *event*

STEP 4 For each row with *event* found do

STEP 4.1 *event* = event listed in "Local Effect(s)" column

STEP 4.2 PL-SFTA_CREATE (*event*)

Figure 35 PL-SFTA_CREATE Algorithm

Following the PL-SFTA_CREATE algorithm, an intermediate node tree is created. Note that this intermediate node tree does not contain any Boolean logic gates, nor does it include any information associating the product line's commonality variability requirements to the hazard. Applying this algorithm for the root node "A spacecraft to asteroid collision occurred" using the SFMECA tables from Section 5.2.1 yields the tree depicted in Figure 36 as one of the subtrees that could potentially cause the root node hazard. Additionally, Figure 37 illustrates a portion of the intermediate node tree for the root node "A spacecraft received solar radiation damage". We will use these as examples to illustrate the steps throughout this section.

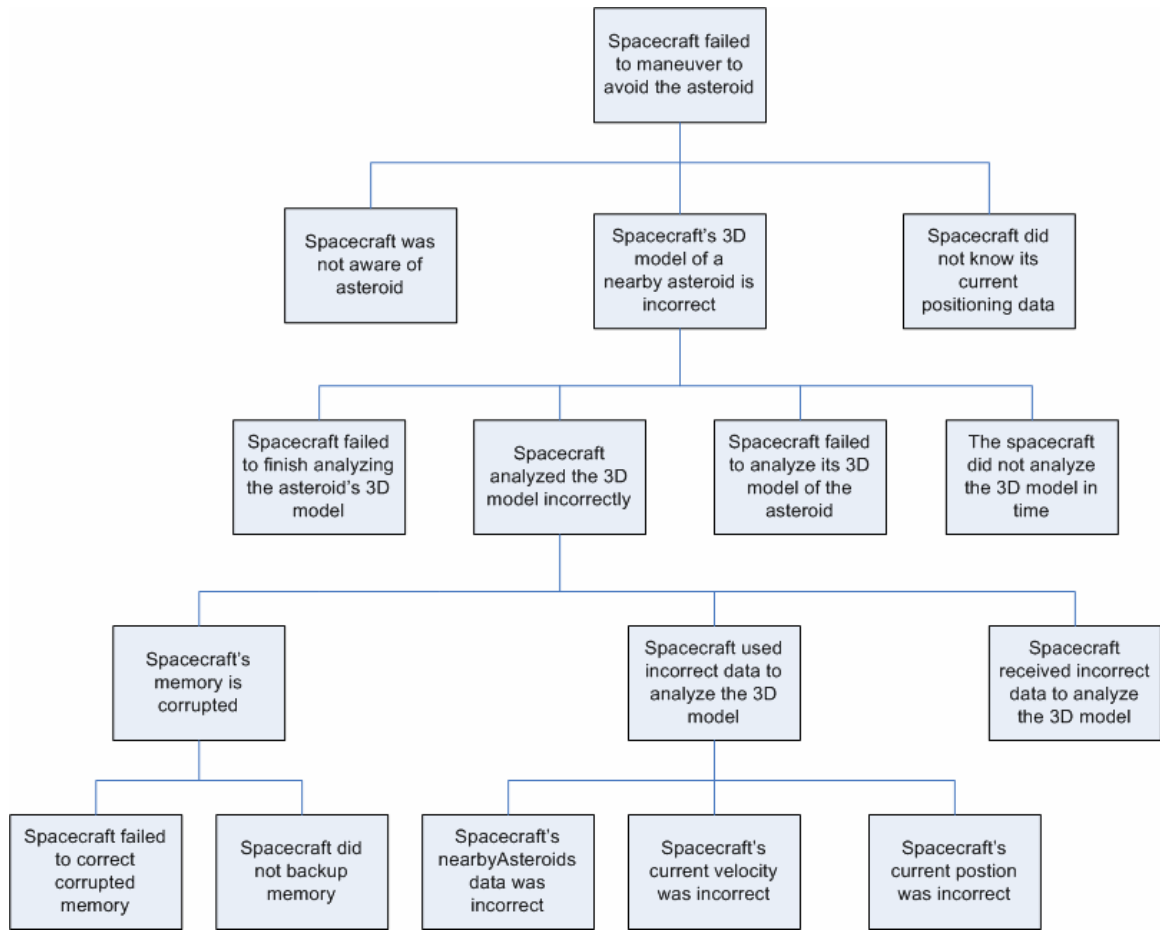


Figure 36 An Excerpt of an Intermediate Node Tree for the Spacecraft to Asteroid Collision Hazard

PLFaultCAT offers no distinct functionality to aid in completion of this step of the product-line software fault tree creation. In fact, PLFaultCAT cannot graphically construct a tree as shown in Figure 36 and Figure 37 without Boolean logic gates relating causal events to the affected events (this is a result from inheriting the software architecture and functionality of the original FaultCAT tool). Rather, the intermediate node tree, constructed manually, acts as an input to PLFaultCAT.

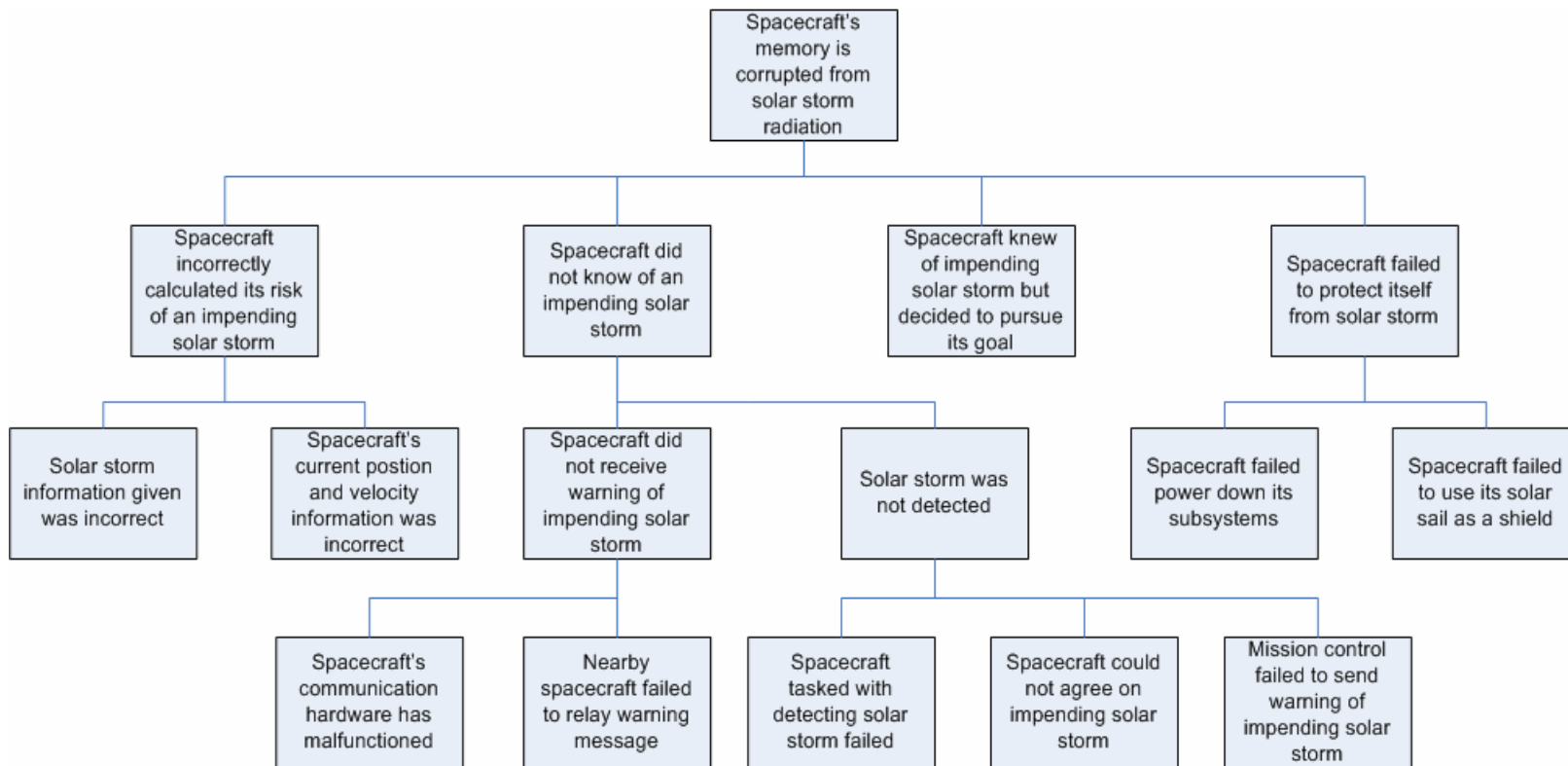


Figure 37 An Excerpt of an Intermediate Node Tree for the Spacecraft Received Solar Storm Damage Hazard

Step 2. Refine the intermediate node tree and input into PLFaultCAT. The intermediate node tree produced in Step 1 can contain nodes that do not reflect the level of detail needed. A single node could actually be the effect of a combination of causes not captured in the SFMECA since a SFMECA typically cannot capture a series of causes leading to a failure event. Thus, domain expertise is needed to analyze the tree for completeness, capture additional events leading to a failure (e.g., events from the environment) and to refine nodes as needed. Using our intermediate node tree shown in Figure 37 for example, it may be desirable to further detail the causes of the node "Spacecraft failed to use its solar sail as a shield" or, if possible, reference a separate fault tree for this failure that details the causal factors.

Depending on the level of detail presented in the SFMECA, it may provide insight into what kind of logic gate should be applied to join children event nodes to their parents. Traditionally, SFMECA only considers a single failure at a time, thus implying logical OR gates throughout a PL-SFTA. This is even more evident when the SFMECA distinguishes the variabilities from each individual failed Item/Event. However, our experience has shown that some detailed SFMECAs provided enough causal information to warrant a logical AND gate. For example, using our SFMECA, shown in Table 4, as well as the intermediate node tree, shown in Figure 36, we can infer that the nodes "Spacecraft failed to correct corrupted memory" and "Spacecraft did not backup memory" must be joined by a logical AND gate in order to cause the "Spacecraft's memory is corrupted" node. Intuitively, this makes sense. Because of the advanced error trappings inherent in a PAM spacecraft, the software will only incur corrupt memory if there indeed has been a memory failure *and* the PAM spacecraft has failed to recognize a memory failure.

The caveat to this approach is that the SFMECA should only be used as a heuristic guide aided by domain knowledge and experts to produce the ultimate logic gate

represented in the PL-SFTA. Thus, the SFMECA should be mined to extract as much relevant information as possible to assist the construction of the PL-SFTA.

Note that a PL-SFTA can be constructed using other methods as input. For example, Leveson asserts that other safety analysis techniques such as a Cause-Consequence Diagram, an Event Tree Analysis, a Hazards and Operability Analysis (HAZOP) and/or a State Machine Model can be used to help guide the construction of a SFTA [44]. We illustrated the use of a SFMECA as a guide to constructing the PL-SFTA since we used it in Section 5.2 and it had not been described in this manner prior to our work in [17].

In addition to refining each node, we apply domain knowledge to determine the necessary logical combination of the children nodes to cause the parent node. This is a similar process to traditional fault tree analysis. Using the PLFaultCAT tool and applying Step 2 to the intermediate node trees found in Figure 36 and Figure 37 yields the intermediate software fault trees depicted in Figure 38 and Figure 39, respectively.

Aside from allowing the user to graphically construct a fault tree, PLFaultCAT allows an annotated description of each node so that the user can attach further details. This is especially advantageous in that it provides traceability to the hazard analysis. It also can be used to cross-check the completeness of the SFMECA by ensuring that all hazard events in the SFTA map to a cause or effect in the SFMECA (i.e., one-way traceability). We illustrated the completeness checking of a SFMECA and a PL-SFTA in Section 5.4 when detailing the Bi-Directional Safety Analysis (BDSA) for a safety-critical multi-agent system product line (MAS-PL) using the SFMECA developed in Section 5.2.

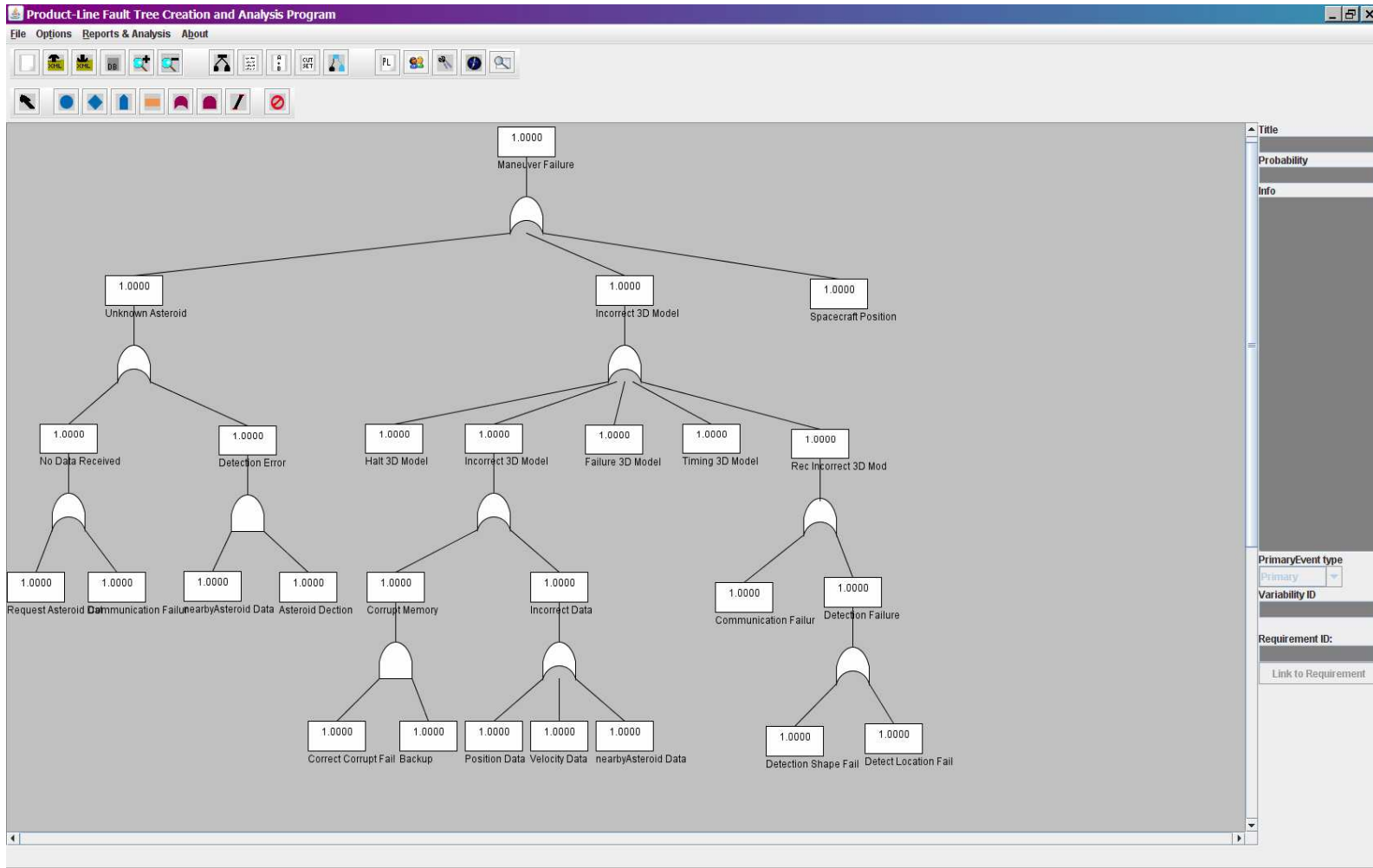


Figure 38 An Intermediate Software Fault Tree for the Spacecraft to Asteroid Collision Hazard in PLFaultCAT

Step 3. Consider the influence of variabilities on all leaf nodes and tag each node accordingly. This is the crux of the product-line construction. In this step we employ a bottom-up approach to analyze each leaf node and determine which commonalities and/or variabilities contribute to causing the root node event to occur. In doing this, we associate the range of commonality and variability choices for any individual product-line member with how it might influence a particular hazard. Not every commonality or variability will have an influence or appear within any given fault tree. However, every leaf event node should have an associated commonality, variability, and/or basic (primary) event (e.g., an environment or user input).

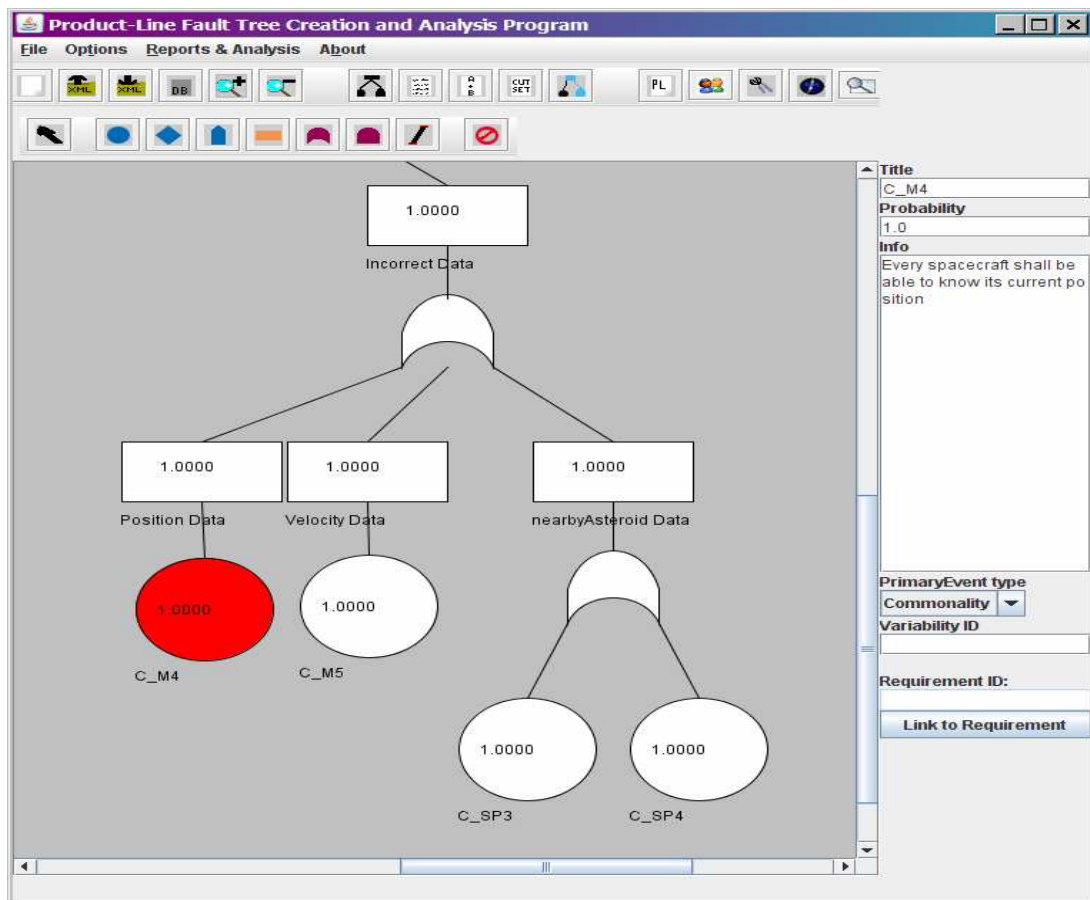


Figure 40 Depicting the Influence of a Commonality for the Spacecraft to Asteroid Collision Fault Tree in PLFaultCAT

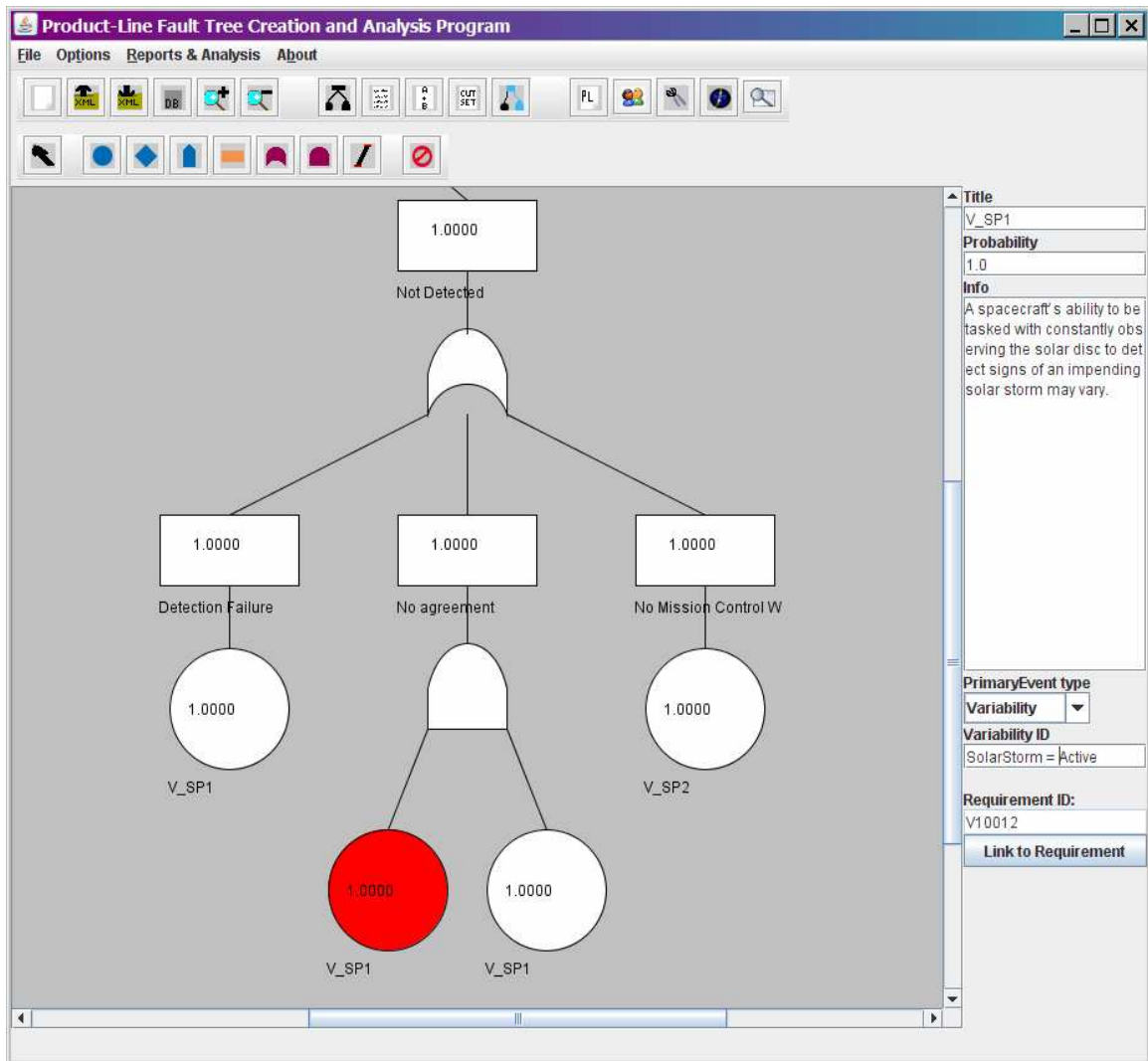


Figure 41 Depicting the Influence of a Variability for the Spacecraft Received Solar Damage Fault Tree in PLFaultCAT

When considering a variability's influence on a particular leaf node, we consider the parameters of variation allowed. While many variabilities are features that are simply present or not present in a product (e.g., a PAM spacecraft will either be able to or not able to receive messages from mission control warning of an impending solar storm, see V_SP2 in Figure 41), some variabilities represent an allowable numerical or enumerated

range for a particular feature (e.g., a PAM spacecraft tasked with monitoring the solar disc to detect an impending solar storm can either assume a passive, warm-spare or active role, see V_SP1 in Figure 41). Considering the influence of a present or absent variability on an event is straightforward; we analyze the influence of the variability being present within the product and not functioning as designed.

If, however, we need to consider an enumerated or range type of variability, we must consider the various possibilities within the variability and their influence on fault tree events. For large ranges, safety analysis on each potential variability choice would be infeasible. Thus, class ranges are used to determine how different ranges could affect contributing events [76].

For example, looking at the node "Detection Failure" in our example, shown in Figure 39, and consulting the CVA, shown in Figure 5 and Figure 6 as well as the Parameters of Variation table given in Table 2, we conclude that this failure node can only occur if the PAM spacecraft has the feature (variability) that it is to constantly monitor the solar disc for impending solar storms using the "active" variation point. Thus, we annotate this node accordingly to indicate that the node "Detection Failure" can only occur when either one of the variabilities (Warm-Spare or Active) is present in a product line member. The representation of this is shown in Figure 41.

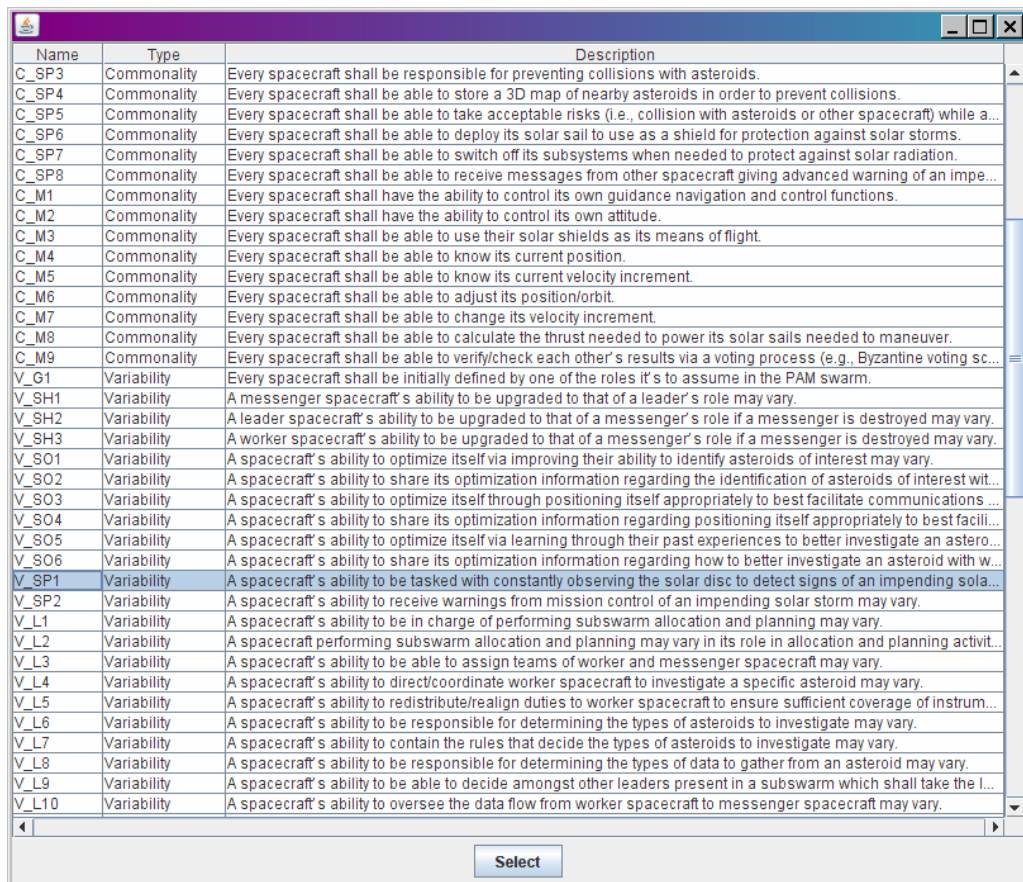
If, however, the node relates to a commonality rather than a variability, we link the fault tree's leaf node with the appropriate commonality. For example, looking at the node "Position Data" in our example, shown in Figure 38, and consulting the CVA, shown in Figure 5 and Figure 6 as well as the Parameters of Variation table given in Table 2, we conclude that this failure node occurs for all PAM spacecraft since all spacecraft will be able to know its position information. Thus, we annotate this node accordingly to indicate that the node "Position Data" failure may occur in every product line member since it is a commonality. The representation of this is shown in Figure 40.

Using PLFaultCAT makes associating a commonality and/or variability with a failure node straightforward. The PLFaultCAT interface allows you to label the "Basic Event" nodes, depicted as circles, as a Commonality (shown in Figure 40 under the heading "PrimaryEvent type") or as a Variability (shown in Figure 41 under the heading "PrimaryEvent type") as well as defining a label or ID for the variability (the textbox under the heading "Variability ID"). In the example, in Figure 41, the variability (feature) has the label "V_SP1" to correspond with the requirement number listed in the CVA. The "Variability ID" describes the variability (feature) so that it will be recognizable later when selecting the variabilities to include in a new product line member. For this example, we simply annotate "SolarStorm = Active" to indicate that a product line member may or may not have this variability (feature).

The consideration of numerical ranges or values is particularly important because often not all values of a variability will contribute to a failure. Applying equivalence class partitioning and boundary value analysis concentrates on the fringe numbers and other frequently error-prone ranges to improve coverage of possible vulnerabilities. Although this situation was not encountered in the PAM case study used in this dissertation, we had encountered it in previous cases. For example, a previous product line had a numerical range variability that the number of sensors that a product may have varies between 1 and 5 sensors. In this case, when we encounter the situation where the variability of multiple wind sensors can cause a failure node and the commonality of having one wind sensor will not, PLFaultCAT can accommodate this case by specifying the variability by labeling it a "Variability" PrimaryEvent Type and specifying in the "Variability ID" field a label indicating that multiple sensors must be present in the product line member to cause the parent failure node. This same approach would be utilized for any enumerated variability.

If the product-line commonalities and variabilities were previously defined using DECIMAL (described in Section 2.1.4 and used within the Gaia-PL methodology in Section 4.2.1.2) [23], [58], [59], the association of a requirement with the leaf node of a fault tree is much easier. Using PLFaultCAT, we can link a fault tree to the DECIMAL XML file for the product line and then select the appropriate requirement using the “Link to Requirement” button. This will bring up a table, shown in

Figure 42, to automatically link the requirement to the leaf node and fill in the details of the requirement in the appropriate text fields.



Name	Type	Description
C_SP3	Commonality	Every spacecraft shall be responsible for preventing collisions with asteroids.
C_SP4	Commonality	Every spacecraft shall be able to store a 3D map of nearby asteroids in order to prevent collisions.
C_SP5	Commonality	Every spacecraft shall be able to take acceptable risks (i.e., collision with asteroids or other spacecraft) while a...
C_SP6	Commonality	Every spacecraft shall be able to deploy its solar sail to use as a shield for protection against solar storms.
C_SP7	Commonality	Every spacecraft shall be able to switch off its subsystems when needed to protect against solar radiation.
C_SP8	Commonality	Every spacecraft shall be able to receive messages from other spacecraft giving advanced warning of an impe...
C_M1	Commonality	Every spacecraft shall have the ability to control its own guidance navigation and control functions.
C_M2	Commonality	Every spacecraft shall have the ability to control its own attitude.
C_M3	Commonality	Every spacecraft shall be able to use their solar shields as its means of flight.
C_M4	Commonality	Every spacecraft shall be able to know its current position.
C_M5	Commonality	Every spacecraft shall be able to know its current velocity increment.
C_M6	Commonality	Every spacecraft shall be able to adjust its position/orbit.
C_M7	Commonality	Every spacecraft shall be able to change its velocity increment.
C_M8	Commonality	Every spacecraft shall be able to calculate the thrust needed to power its solar sails needed to maneuver.
C_M9	Commonality	Every spacecraft shall be able to verify/check each other's results via a voting process (e.g., Byzantine voting sc...
V_G1	Variability	Every spacecraft shall be initially defined by one of the roles it's to assume in the PAM swarm.
V_SH1	Variability	A messenger spacecraft's ability to be upgraded to that of a leader's role may vary.
V_SH2	Variability	A leader spacecraft's ability to be upgraded to that of a messenger's role if a messenger is destroyed may vary.
V_SH3	Variability	A worker spacecraft's ability to be upgraded to that of a messenger's role if a messenger is destroyed may vary.
V_SO1	Variability	A spacecraft's ability to optimize itself via improving their ability to identify asteroids of interest may vary.
V_SO2	Variability	A spacecraft's ability to share its optimization information regarding the identification of asteroids of interest wit...
V_SO3	Variability	A spacecraft's ability to optimize itself through positioning itself appropriately to best facilitate communications ...
V_SO4	Variability	A spacecraft's ability to share its optimization information regarding positioning itself appropriately to best facili...
V_SO5	Variability	A spacecraft's ability to optimize itself via learning through their past experiences to better investigate an astero...
V_SO6	Variability	A spacecraft's ability to share its optimization information regarding how to better investigate an asteroid with w...
V_SP1	Variability	A spacecraft's ability to be tasked with constantly observing the solar disc to detect signs of an impending sola...
V_SP2	Variability	A spacecraft's ability to receive warnings from mission control of an impending solar storm may vary.
V_L1	Variability	A spacecraft's ability to be in charge of performing subswarm allocation and planning may vary.
V_L2	Variability	A spacecraft performing subswarm allocation and planning may vary in its role in allocation and planning activit...
V_L3	Variability	A spacecraft's ability to be able to assign teams of worker and messenger spacecraft may vary.
V_L4	Variability	A spacecraft's ability to direct/coordinate worker spacecraft to investigate a specific asteroid may vary.
V_L5	Variability	A spacecraft's ability to redistribute/realign duties to worker spacecraft to ensure sufficient coverage of instrum...
V_L6	Variability	A spacecraft's ability to be responsible for determining the types of asteroids to investigate may vary.
V_L7	Variability	A spacecraft's ability to contain the rules that decide the types of asteroids to investigate may vary.
V_L8	Variability	A spacecraft's ability to be responsible for determining the types of data to gather from an asteroid may vary.
V_L9	Variability	A spacecraft's ability to be able to decide amongst other leaders present in a subswarm which shall take the l...
V_L10	Variability	A spacecraft's ability to oversee the data flow from worker spacecraft to messenger spacecraft may vary.

Figure 42 Automatically Linking a Product-Line Requirement to a Software Fault Tree Node in PLFaultCAT

5.3.3.3 Discussion

Throughout the development and construction of the product-line Software Fault Tree Analysis (PL-SFTA) we associate commonalities and/or variabilities with each leaf node in the intermediate node tree developed in Step 2 of Section 5.3.3.2. This process may yield both a commonality and variability being associated with a single failure node. In this case, intuition may suggest disregarding consideration of the variability since the causal event will always be present due to the presence of the associated commonality node. However, the risk of failure posed by the commonality may be mitigated while the risk posed by the variability remains. Hence, the variability must be retained to aid in the analysis of the product line, especially as the product line evolves.

Neither the construction of a PL-SFTA nor PLFaultCAT captures product-line dependencies. This is because the role of the product-line SFTA is to give as complete an account as possible of potential contributing causes to the root node. Note that the PL-SFTA does not enforce existing product-line dependencies. Instead, it represents all possible permutations of choices of values of product-line members and relies on dependency enforcement prior to the application engineering phase. We discuss this issue using DECIMAL as a tool to enforce product-line dependencies in Section 5.3.5.

Since SFTA adopts a slightly different perspective when viewing the product line, it is not uncommon to discover missing requirements. The construction of the product-line SFTA in PLFaultCAT may have some feedback effect on the CVA in terms of discovering previously unidentified dependencies. Similarly, missing commonalities and variabilities, or incorrect parameters of variation may sometimes be identified via this process. We discuss this issue using the automated safety analysis tools in PLFaultCAT in Section 5.3.4.

It is interesting to note that the influence of variabilities on hazards will not necessarily “sink to the bottom” of the fault tree but can instead be dispersed throughout

the tree. Variabilities are commonly thought of as refinements of commonalities so the expectation is that they will only influence the root node from the lowest levels of the fault tree [49]. However, we found that this was not always the case. Variabilities, especially in software, are sometimes add-on features to the system rather than refinements of a commonality. Feature-oriented variabilities can spawn refinement variabilities of their own. Situations like this can lead to a PL-SFTA where variabilities are spread throughout the levels of the tree rather than clustered at the bottom.

It is important to note that the method outlined in Steps 1-3 of Section 5.3.3.2 is an iterative process that is repeated for all hazards in the hazards list. This will produce a set of product-line software fault trees.

5.3.3.4 Creating the PL-SFTA for the PAM Case Study Examples

Applying this step to the Prospecting Asteroid Mission (PAM) case study used in this dissertation for the “A spacecraft to asteroid collision occurred” root node, discussed in Section 5.3.3.2 yielded a 143-node product line Software Fault Tree Analysis (PL-SFTA), as partially shown in Figure 43. Specifically, the fault tree consisted of 82 failure nodes and 61 commonality/variability nodes (of which 54 were for product-line commonalities and 7 were for product-line variabilities). Similarly, applying this step for the “A spacecraft received solar radiation damage” root node, discussed in Section 5.3.3.2 yielded a 137-node PL-SFTA. Specifically, the fault tree consisted of 87 failure nodes and 50 commonality/variability nodes (of which 30 were for product-line commonalities and 20 were for product-line variabilities). In these two cases, for example, approximately 76% of the product-line requirements associated to the leaf nodes of the fault trees were product-line commonalities (i.e., leaf nodes that will be found in all configurations of the PAM spacecraft for this particular hazard). A full set of the PL-SFTA hazards for the PAM case study are given in Appendix F.

5.3.4 Deriving Additional Safety Requirements from the Product-Line Software Fault Tree Analysis

After the creation of a product-line software fault tree analysis (PL-SFTA) for a safety critical product line, PLFaultCAT provides software engineers with the opportunity to further analyze the system for safety. Figure 33 provides an overview of the safety analyses possible using a PL-SFTA created in PLFaultCAT. In this section, we detail how new product-line safety requirements can be derived from the PL-SFTA, how PLFaultCAT can automatically identify single-point failures and how PLFaultCAT can identify safety-critical requirements and safety-critical interactions. We also discuss other analyses that can contribute to the safety analysis of a product line.

5.3.4.1 Deriving New Product-Line Constraints

The product-line Software Fault Tree Analysis (PL-SFTA), described in Section 5.3.3, can aid in the discovery of latent safety requirements by identifying high-risk variabilities and common causes and by identifying new constraints. The PL-SFTA construction process produces a set of fault trees with the corresponding contributing commonalities and variabilities attached to the appropriate leaf nodes. Using this set of software fault trees, we can identify or even tabulate the most frequent variabilities that contribute to the root node hazards. If certain variabilities contribute to root node hazards, additional safety requirements and/or hazard analysis may be warranted to mitigate their contribution to hazard nodes.

Any high-level event node within a PL-SFTA that has two or more variabilities connected by an AND gate may warrant a new constraint. Introducing a new product-line constraint limiting the variability combinations in this situation can preclude occurrence of this event node and potentially rid the PL-SFTA from this hazard altogether. However,

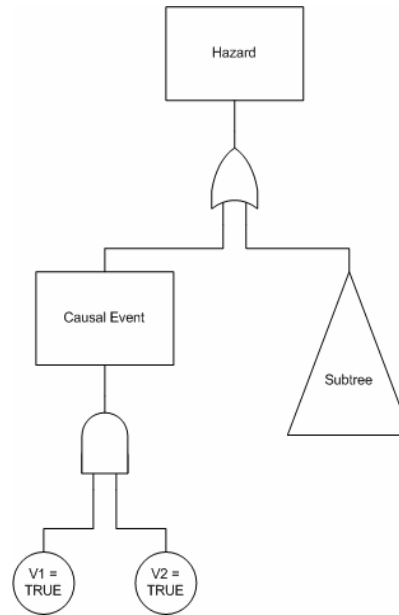


Figure 45 A Generic Product-Line Software Fault Tree Analysis

care must be taken in deriving new product-line dependencies so that the product line is not too limited. The hazard severity as well as the existence of alternative preventive measures must be weighed against the addition of product-line dependencies.

Figure 45 shows a generic example of the derivation of a new product-line constraint from a logical AND gate connecting two variabilities. This example shows that we can mitigate the "Causal Event" node by restricting a system in the product line from having both V_1 and V_2 features. If this is found to be an acceptable solution, the PL-SFTA then retains the "Causal Event" subtree for completeness, but the occurrence of the subtree becomes essentially impossible.

Imposing additional safety requirements in the domain engineering phase improves the product-line specifications and reduces rework in the application engineering phase. The safety requirements and/or product-line dependencies derived from the PL-SFTA can reduce the analysis needed and reduce time-to-market for new products.

5.3.4.2 Identifying Single-Point Failures In PLFaultCAT

An advantage of a traditional Software Fault Tree Analysis (SFTA) as a safety analysis technique is the ability to quickly determine the presence of single-point failures of a single system (i.e., a root node in the SFTA followed by a logical OR gate). A product-line SFTA (PL-SFTA) allows for the quick identification of single-point failures over the entire product-line. An example product-line single-point failure for the Prospecting Asteroid Mission (PAM) case study is shown in Figure 38 and Figure 39.

In the case of a multi-agent system product line (MAS-PL), identifying a single-point failure in a PL-SFTA provides the ability to pinpoint possible single-point failures for every possible instantiation of an agent (adhering to the commonalities and variation points allowed within the MAS-PL). This is advantageous over the traditional application of a SFTA to a MAS-PL because it is not necessary to manually create each SFTA for each possible instantiation of an agent and then manually inspect each SFTA for a single-point failure.

To aid in the identification of single-point failures of a PL-SFTA, PLFaultCAT provides a single-point failure analysis to automate this process. A *single-point failure analysis* searches the set of fault trees in a PL-SFTA for single-point failures (i.e., those hazards connected by a logical OR gate in the SFTA) at user-specified depth, as shown in Figure 46. Since a safety-critical product line will typically have several, large fault trees (e.g., the PL-SFTA for only two of the fault trees for the PAM case study had nearly 250 nodes), the automation of identifying single-point failure will lessen the burden placed on a safety engineer to manually go through this process. In most cases, a safety engineer may only be interested in a single-point failure for the causal events directly leading to the root node failure (i.e., level 1 in PLFaultCAT), PLFaultCAT allows the user to supply the depth for situations when a deeper analysis is desired or required.

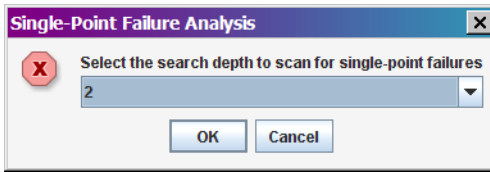


Figure 46 Selecting the Depth to Search for the Single-Point Failures of a PL-SFTA

PLFaultCAT searches the set of fault trees of a PL-SFTA for a product line and provides the details of the discovered single-point failures, as shown in Figure 47, for the “Spacecraft to Asteroid Collision” root node, to aid in developing a new safety requirement (if needed) to mitigate against the single-point failure(s). Upon the identification of a single-point failure, engineers have the opportunity to take mitigating steps to improve the safety, dependability and/or reliability of the system. Further, since SFTA is constructed early in the development lifecycle, mitigating steps can be taken quite early in software product line’s design and development.

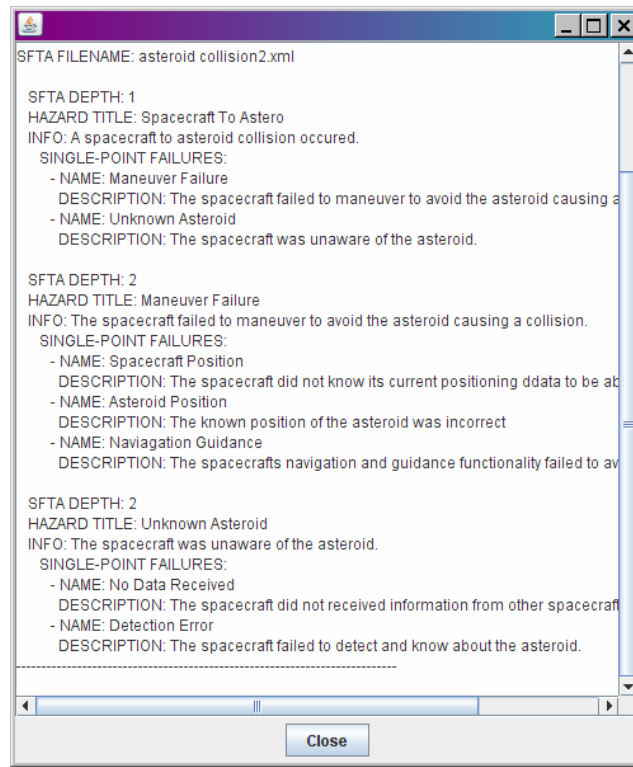


Figure 47 The Single-Point Failure Report Produced by PLFaultCAT

The mitigation of a single-point failure within the PL-SFTA can be done by introducing additional requirements, architectural components, guard conditions, operating rules or other counteractions into the design. For example, using PLFaultCAT to identify the single-point failures for the “Spacecraft to Asteroid Collision” root node of the PAM case study, shown in Figure 47, a mitigation requirement can be introduced to turn the subtree rooted at the “Maneuver Failure” into a non-single-point failure, shown in Figure 43. This node represents the event that spacecraft’s actions to maneuver itself to prevent a collision with the asteroid did not suffice. Using the information provided in PLFaultCAT’s single-point failure report, shown in Figure 47, we can understand that there are three contributing events that can cause this event: 1. the spacecraft’s position data (i.e., current position, current velocity, current orbit, etc.) may be incorrect; 2. the spacecraft’s data on the asteroid (i.e., position, shape, 3D model, gravitational field, etc.) may be incorrect; and 3. the spacecraft’s navigation and guidance functionality (i.e., calculating the thrust needed, utilizing the solar sail for navigation, etc.) may have failed. Each of these events contributes to the spacecraft devising a course that fails to maneuver away from the asteroid to prevent a collision. Thus, the action that a spacecraft takes to prevent a collision with an asteroid is individual. If however, a requirement is introduced to oblige a spacecraft to get a confirmation of its planned course of action to avoid the asteroid (i.e., an independent spacecraft to devise a course based on its data of the requesting spacecraft’s positioning data, asteroid data and navigation and guidance functionality). Using this approach, the spacecraft and another spacecraft would independently and redundantly calculate how to avoid an asteroid and both spacecraft’s calculations would have to fail for the “Maneuver Failure” node rooted at this subtree to occur. Introducing this requirement into the PAM MAS-PL’s Commonality and Variability Analysis (CVA) and updating the PL-SFTA accordingly will yield the fault tree shown in Figure 48.

Note that the uppermost single-point failure (i.e., level 1 in Figure 47) could not be mitigated against since an asteroid has the ability to assume some risks of a collision with an asteroid if the potential scientific outcome outweighs the risk (see requirement C_SP5 in the CVA in Appendix A, page 235). Thus, the collision of a spacecraft and an asteroid is not always a hazard that can be prevented.

5.3.4.3 Identifying Safety-Critical Requirements in PLFaultCAT

Using a product-line Software Fault Tree Analysis (PL-SFTA) for a safety-critical product line also allows for the identification of product-line variability requirements or combinations of variability requirements that disproportionately contribute to hazards. Scanning the leaf nodes (where commonalities and variation points are associated to low-level hazards) of the PL-SFTA can lead to the discovery that a particular variability or set of variation points can contribute to high number of hazards within the set of fault trees of a PL-SFTA. This information proves valuable if engineers determine that the hazard risk of leaving the product line's design unchanged is unacceptable from a safety, dependability and/or reliability standpoint.

The *variability failure contribution analysis* in PLFaultCAT analyzes all the PL-SFTAs to find those variabilities or combination of variabilities that contribute to a high number of hazards. PLFaultCAT performs this analysis and provides a user with an ordered list of the most frequently cited product-line commonality and variability in the set of fault trees of a PL-SFTA, as shown in Figure 49. The requirements value tries to measure its impact on the leaf node failures in the PL-SFTA. For each leaf node where a requirement is the sole requirement or the requirement forms a disjunction with other requirements (i.e., joined by a Boolean OR gate), the requirements value is increased by one since it solely can contribute to the leaf node failure. If, however, the requirement forms a conjunction with other requirements (i.e., joined by a Boolean AND gate)

associated to a leaf node, each requirement's value is increased by its proportion of the conjunction (i.e., one divided by the number of requirements in the conjunction). In addition, the variability failure contribution report provides a listing of the groups of requirements that were found to contribute to a high number of leaf node hazards. For example, in the PAM case study, it was found that the requirements related to a spacecraft correctly knowing its positioning information (i.e., current location, current velocity, current orbit, etc.) as well as having an accurate 3D model of nearby asteroids are the most safety-critical requirements related to the "Spacecraft to Asteroid Collision" fault tree.

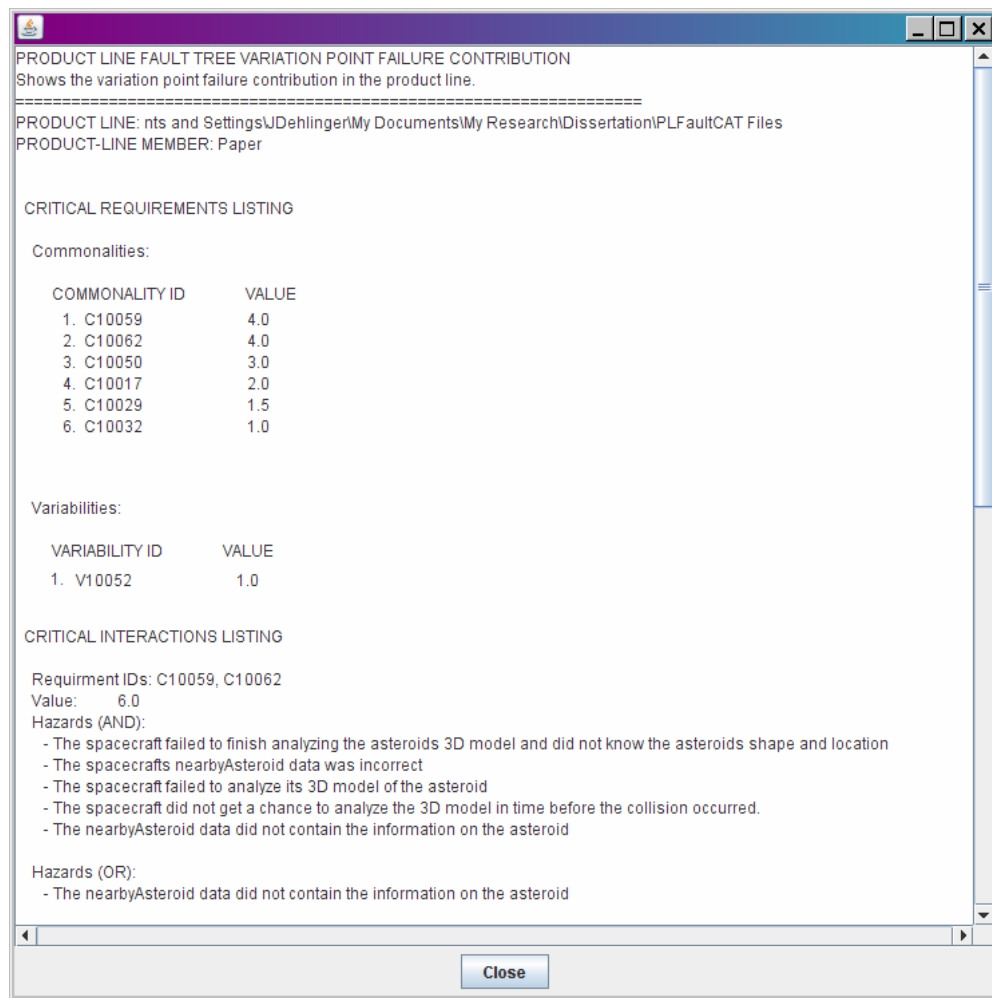


Figure 49 The Variability Contribution Failure Report Produced by PLFaultCAT

A mitigation strategy for combinations of product-line variabilities can be to add dependencies, as described in Section 5.3.4.3. For those combinations of product-line variabilities that contribute to a disproportionately high number of leaf node failures, it may be necessary to restrict the combinations of these features, via product-line dependencies, to achieve safety. The inclusion of product-line dependencies in this case would preclude a product-line member from having this combination of features and prevent the possible leaf node hazards caused by the interaction of these features modeled in the fault tree.

Alternatively, a similar strategy for mitigating single point failures can be adopted, as described above, to limit the impact that the combinations of these variabilities can have on the safety of the system.

This analysis, in particular, has been found to be useful to help guide another safety analysis technique, briefly discussed in Section 5.3.8, that investigates feature interactions. Although a PL-SFTA models a failure statically, a combination of requirements that leads to a disproportionately high number of leaf node failures may act as a guide for a dynamic approach to further investigate the interaction of the requirements' behaviors. In our experience, the *variability failure contribution analysis* helps to scope the feature interactions that should be examined more deeply.

5.3.4.4 Additional Safety Analyses in PLFaultCAT

In addition to the single-point failure analysis and the variability failure contribution analysis, PLFaultCAT provides additional analysis tools that may be useful when assessing the safety or reliability of a product line. The *minimum-cut set analysis* analyzes a single fault tree of a PL-SFTA and identifies the smallest sets of events that must occur such that the root node accident will occur. The *probability report* calculates the probability of occurrence of the root node given the probabilities of all other nodes.

Although this may not apply for software since failure probabilities for software are difficult to obtain, this calculation may be helpful when hardware is included in a PL-SFTA.

5.3.5 Reusing the Product-Line Software Fault Tree Analysis to Derive the Software Fault Tree Analysis for a Product-Line Member

In Section 5.3.3, we detailed the construction of a product-line Software Fault Tree Analysis (PL-SFTA) in PLFaultCAT. In this section, we describe the reuse of the PL-SFTA to derive the software fault trees for new product-line members. The derivation of the software fault trees for new product-line members from the PL-SFTA occurs during the application engineering phase of the Weiss and Lai’s Family-Oriented Abstraction, Specification, and Translation (FAST) model [88]. In the Gaia-PL methodology, shown in Figure 30, the derivation of the software fault trees for new agent of a multi-agent system product line (MAS-PL) from the PL-SFTA occurs during the Analysis and Design Phase.

In this section, we first describe how to define a new product-line member within the context of the previously defined product-line commonalities and variabilities. Then we describe how to prune the PL-SFTA, aided by PLFaultCAT, so that the previously performed safety analysis is be reused. Finally, this section also discusses the flexibility of the product-line SFTA in supporting product-line evolution as well as limits on reuse.

5.3.5.1 Pruning the PL-SFTA for a New Product-Line Member

In product-line Software Fault Tree Analysis (PL-SFTA) we use a pruning process followed by a structured inquiry to develop a new product-line member’s Software Fault Tree Analysis (SFTA) from the PL-SFTA. Figure 43 shows a PL-SFTA for the Prospecting Asteroid Mission (PAM) case study for the “A spacecraft to asteroid collision occurred” root node. The reuse of the PL-SFTA performed using PLFaultCAT

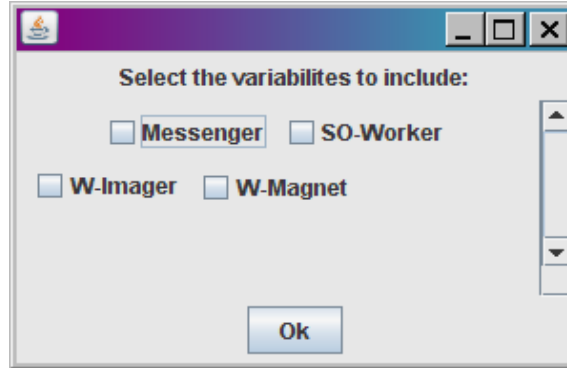


Figure 50 The Variability Selection Window to Prune a PL-SFTA

for a new system in the product line has three basic steps: selecting the variabilities for a new product line member, deriving the product-line member SFTA and applying domain knowledge, each of which are described below.

Step 1. Select the variabilities for new the product-line member. Producing a product-line member entails a selection of which variabilities or features to include. This process can include an ordering of variability selection (e.g., according to domain model techniques in [88]) or can leave the selection process to the system engineers.

PLFaultCAT facilitates the selection of product-line member's variabilities through a checkbox window that presents all possible variabilities, shown in Figure 50 for some of the variation points possible in the PAM case study.

A product-line member is created by selecting the variabilities that it will contain and defining the values of the variabilities. Typically, the selection of a set of variabilities does not guarantee a legal product-line member. Rather, the choice of variabilities must satisfy the previously established product-line dependencies and constraints. Thus, verification must then show that the set of variabilities do not violate any of the defined product-line dependencies. This is an easy verification check to perform manually on a

small product line but requires automated support as the number of variabilities and dependencies increase.

PLFaultCAT does not enforce nor check the dependencies prescribed in the Commonality and Variability Analysis (CVA). Instead, other tools are capable of enforcing the dependencies and constraints detailed in the CVA for large, complex product lines [58], [59]. PLFaultCAT is used after the choice of variabilities has been determined to be legal.

To assist in the definition of a product-line member, DECIMAL (described in Section 2.1.4 and used within the Gaia-PL methodology in Section 4.2.1.2) [23], [58], [59] allows a user the flexibility to select the variabilities for a new product-line member and then define the values for each variability. To make this approach scalable, DECIMAL *automatically* verifies that the proposed new product-line member's set of variabilities does not violate the defined dependencies. Further, DECIMAL verifies that all values of the variabilities fall within the allowed ranges. If any violations are discovered, DECIMAL flags them so that the developer can rectify the problem.

If DECIMAL is used in this manner, PLFaultCAT can read the verified product-line member(s) from DECIMAL and *automatically* derive the product-line members' SFTAs from the PL-SFTA, as described in Step 2. Using this approach, the user chooses the verified product-line members within PLFaultCAT rather than the variabilities to include, as described above, before PLFaultCAT will prune the PL-SFTA to derive the product-line members' fault trees.

For illustration purposes, we consider a *leader* PAM spacecraft for the fault tree with a root node of "Spacecraft to Asteroid Collision". The variation points of this spacecraft are not all included in this particular fault tree since the *leader* PAM spacecraft are not tasked with detecting, reporting or archiving the 3D model of an asteroid, an important variability requirement associated to many of the leaf nodes in the fault tree

(see Section 5.3.4.3). Thus, the resulting fault tree using the pruning algorithm described in Step 2 should only include those parts of the fault tree that are associated to product-line requirements. The resulting pruned fault tree, for this spacecraft, also represents the core of the fault tree that will be present for any members of the PAM MAS-PL and provides a measurement of the reuse potential of the PL-SFTA.

Step 2. Derive the product-line member SFTA. After establishing and verifying a product-line member, we prune the product-line SFTA to create a baseline SFTA for the new system. The pruning process first uses a depth-first search to automatically remove the subtrees that have no impact on the product-line member being considered and then relies on a small amount of domain knowledge to further collapse and prune the SFTA. For each verified product-line member, the algorithm starts with the root node as *node* and proceeds as follows:

PL-SFTA_SEARCH (*node*):

STEP 1 If *node* is not a commonality or a selected variability then

STEP 1.1 Perform DFS for a selected variability or commonality node

STEP 1.2 If DFS returns true

STEP 1.2.1 For each child node do

STEP 1.2.1.1 PL-SFTA_SEARCH(*node*)

STEP 1.3 If search returns false then

STEP 1.3.1 Remove subtree rooted at *node*

STEP 2 Else if *node* is an unselected variability then

STEP 2.1 Remove subtree rooted at *node*

A “selected variability” in our algorithm is an optional feature that is required in the new system. That is, it is a variability requirement that has been included in the

definition of a new product-line member. For example, a select set of *worker* PAM spacecraft equipped with a visible imager may be tasked to detect the size, shape and location of an asteroid and construct a 3D model of the asteroid so that other PAM spacecraft can use the 3D model to avoid a collision. An unselected variability, however, is an optional feature or a value of a variability not present in the new system.

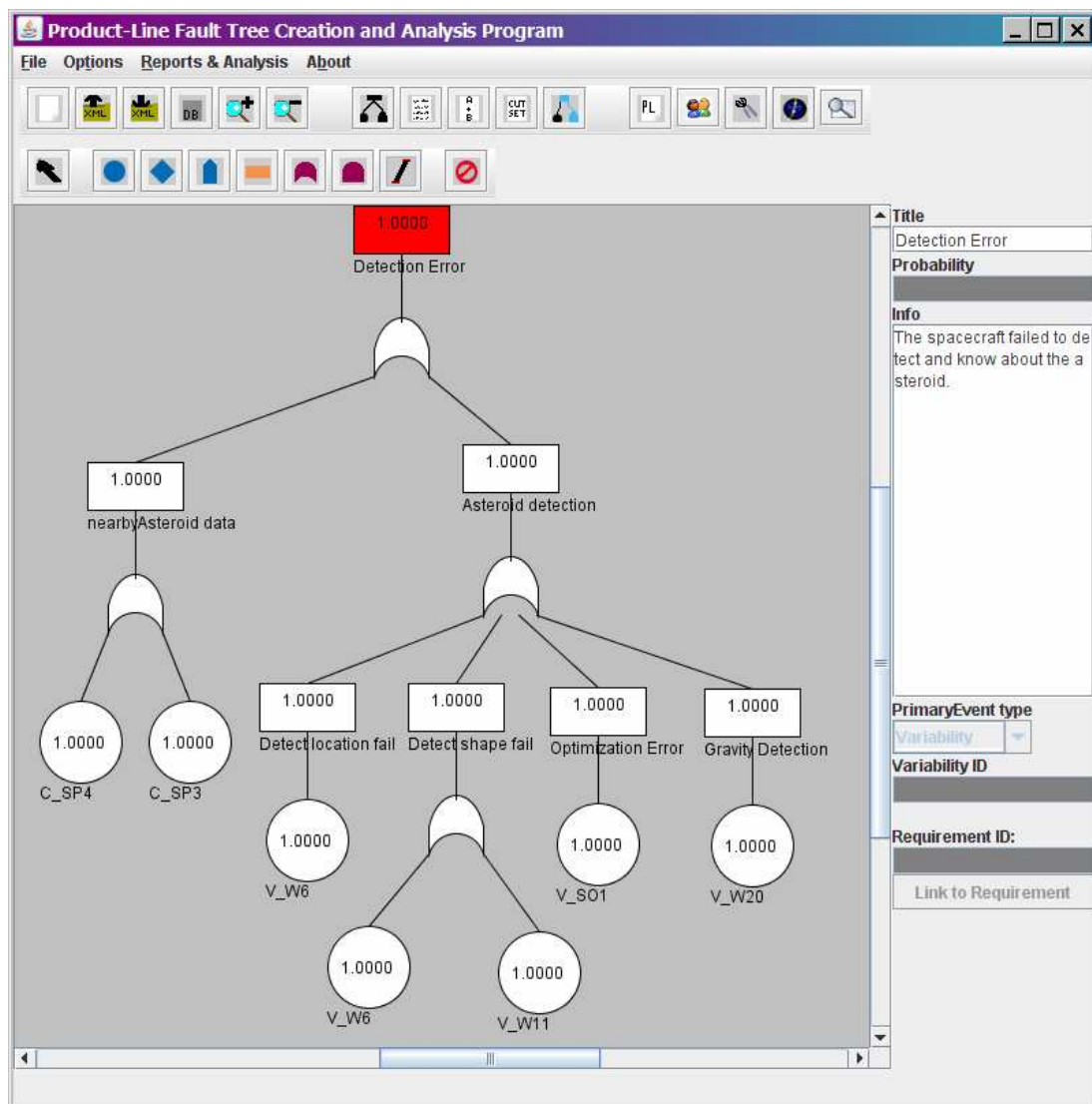


Figure 51 Pruning the PL-SFTA in PLFaultCAT

With multiple SFTAs and many nodes in each SFTA, this pruning is not scalable or practical in an industrial setting without such tool support. PLFaultCAT implements this algorithm using the variabilities specified to include in the product-line member, as in Step 1. The tool processes the PL-SFTA XML file to create a new fault tree including only those nodes associated with the commonalities and chosen variabilities for the new system. In Step 3 of the domain engineering phase, described in Section 5.3.3.2, a label was attached to every variability by giving a variability name in the "Variability ID" textbox. Alternatively, the leaf node could have been directly associated to a variability defined in DECIMAL, as shown in Figure 42, using PLFaultCAT's "Link to Requirement" button automatically filling in the required information into the "Variability ID" textbox. It is this label, for the chosen variabilities, that is searched for in the XML file to decide whether a variability node should be retained. Upon completion of the PL-SFTA_SEARCH logic implemented in PLFaultCAT, the set of fault trees for the new product-line members are stored in an XML format.

The subtree of the "Spacecraft to Asteroid Collision" fault tree shown in Figure 51 illustrates how the pruning algorithm executes within PLFaultCAT to remove irrelevant subtrees. Using the PL-SFTA_SEARCH algorithm for a PAM *leader* spacecraft, we see that the subtree rooted at "Asteroid Detection" contains neither a failure node associated with a commonality or with a selected variability. Intuitively, this implies that this particular spacecraft does not have any functionality related to the detection of the characteristics of an asteroid, which is true in this case. Thus, the PL-SFTA_SEARCH algorithm used in PLFaultCAT will remove this entire subtree since it can have no influence on any of the parent failure nodes of this subtree, shown in Figure 52. If, however, we consider a PAM *worker* spacecraft equipped with a visible imager and tasked with detecting the shape characteristics of an asteroid for the subtree

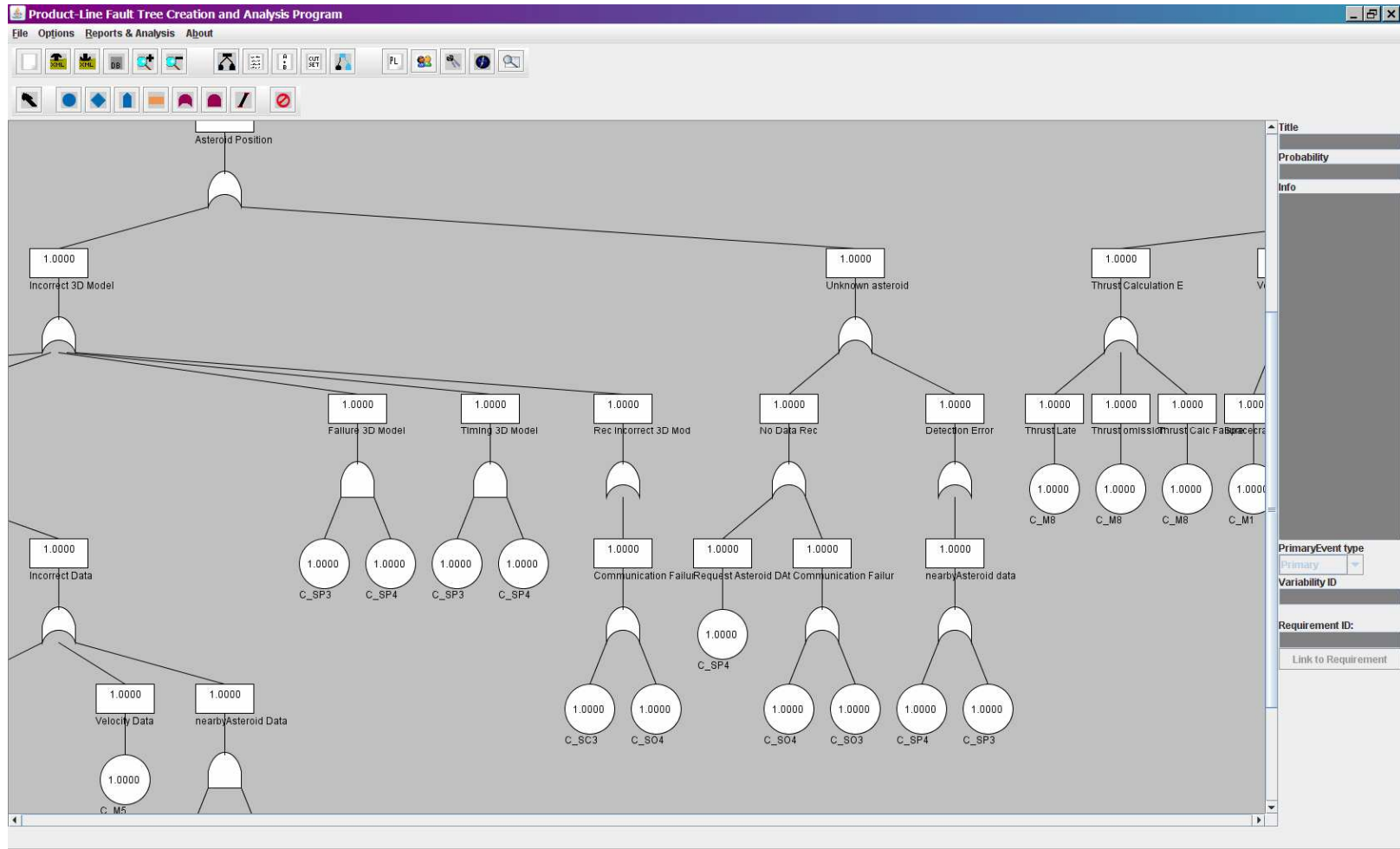


Figure 52 Pruning for a Product-Line Member Software Fault Tree in PLFaultCAT

illustrated in Figure 51, we see that the entire subtree should be retained since the selected variabilities all can have an influence on the failure nodes in each path of the subtree. PLFaultCAT uses this logic in a depth-first fashion over the entire PL-SFTA to derive the product-line member's fault tree based on the selected variabilities.

For example, using the 143-node PL-SFTA constructed in Section 5.3.3 on the hazard “A spacecraft to asteroid collision occurred”, PLFaultCAT was used to automatically derive the fault trees for a PAM *leader* spacecraft that has no functionality to detect the shape characteristics of an asteroid, as described above. The initial execution of PLFaultCAT reduced the number of failure nodes by approximately to 69 of the original 82 nodes (i.e., approximately 16%) and is partially shown in Figure 52.

The PL-SFTA_SEARCH algorithm errs on the side of caution since it only marks the subtrees that can be removed without review and does not actually do any pruning. This is advantageous from a safety perspective because the application of the algorithm simply indicates those subtrees where neither commonalities nor selected variabilities can be found in the subsequent children nodes. This algorithm then defers the actual pruning to the domain experts, as described in the next step.

3. Apply domain knowledge. After removing the subtrees that had no bearing on the product-line member under consideration, the tree may be able to be further pruned and/or collapsed within PLFaultCAT. However, this step requires domain knowledge. This also illustrates the limit to completely automated product-line Software Fault Tree Analysis (PL-SFTA) reuse. Removal of subtrees will often lead to orphaned logic gates or other opportunities to safely simplify the fault trees of a new product-line member, shown in Figure 52 for the subtree rooted at “Detection Error” discussed in Step 2.

Collapsing orphaned OR gates are trivial. If there is only one causal event remaining, we collapse the lower event into the parent event. If there is only one commonality or variability leaf node remaining, we attach it to the parent event and

remove the OR gate. When AND gates are involved, we need to be more cautious. Intuitively, if at least one input line to an AND gate is removed, the output event is impossible. However, it was found that this is not always the case and thus each removal of an AND gate warrants further scrutiny.

The clean up of the derived fault trees for the new product-line member(s) presented in this step is a manual process and must be pursued with utmost care. Enough information should be retained within the product-line member's fault tree to provide ample information for future hazard analysis and mitigation strategies. It is in this light that the subtree shown in Figure 52 reduces to the subtree shown in Figure 53 by removing the useless logic gates and connecting the failure nodes. In this example, manual pruning further reduced the number of nodes for a PAM *leader* spacecraft for the “A spacecraft to asteroid collision occurred” fault tree to 64 of the original 82 nodes.

The application of domain knowledge to the fault tree resulting from Step 2a is beneficial step in the derivation of the fault trees for a new product-line member(s) because it removes the extraneous nodes and focuses attention on nodes that can potentially contribute to failures in a specific product-line member.

Note that, as shown in Figure 33, the set of fault trees for the new product-line members can additionally utilize the safety analyses provided by PLFaultCAT, detailed in Section 5.3.4 to further verify the safety of the product-line member's SFTA. After the pruning of the PL-SFTA to derive the fault trees for a new product-line member, it may be the case that single-point failures that were not present in the PL-SFTA are now present a specific product. Thus, further safety analysis of the SFTA for a product-line member should be considered. Since the SFTA for a product-line member is simply a traditional SFTA, traditional software safety analysis techniques not covered in this dissertation may be useful.

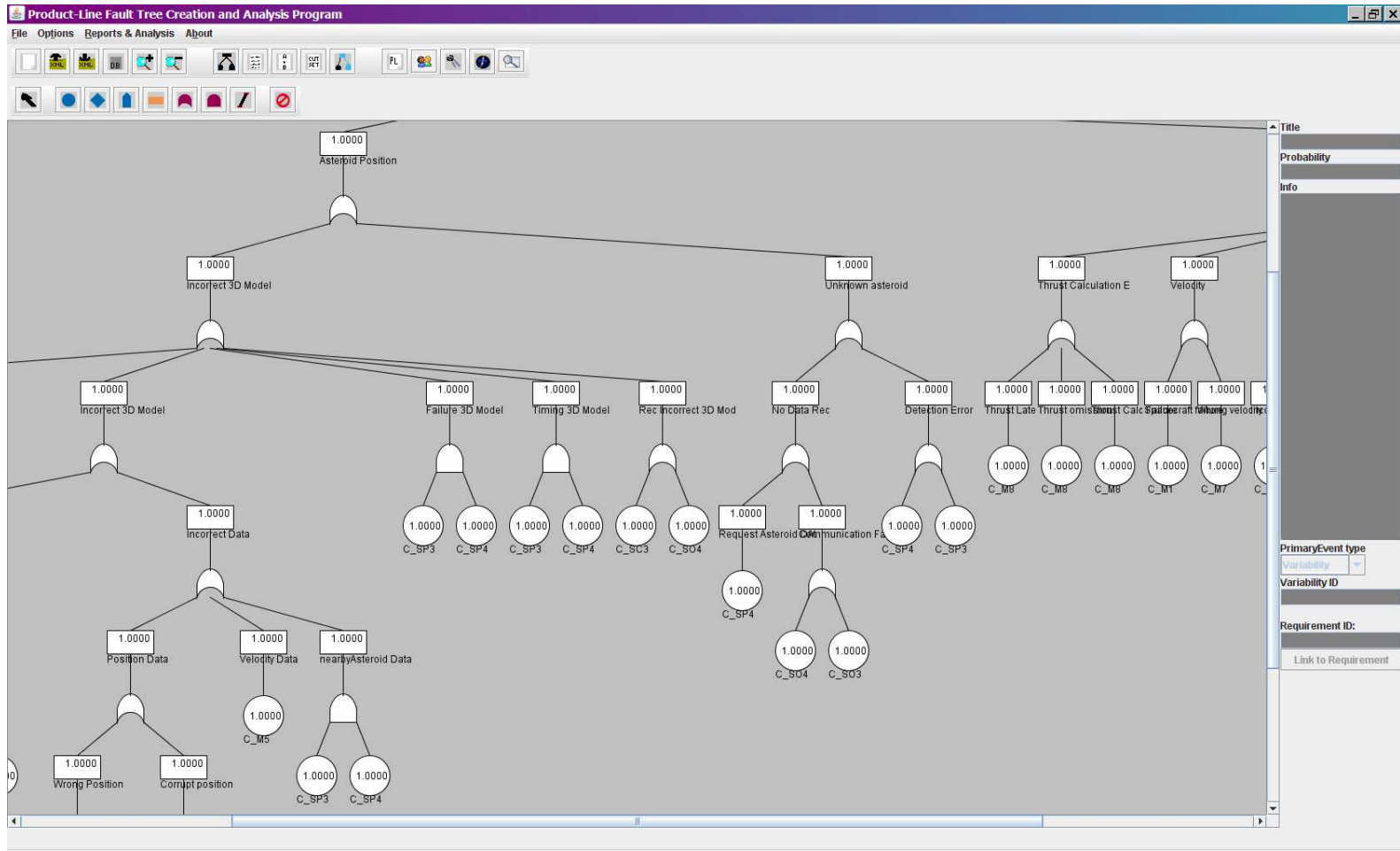


Figure 53 Resulting Product-Line Member Software Fault Tree after Manual Pruning

5.3.5.2 Pruning the PL-SFTA for the PAM Case Study Examples

Looking back, the “A spacecraft to asteroid collision occurred” fault tree for a Prospecting Asteroid Mission (PAM) *leader* spacecraft, the clean up process described in Step 3 removed an additional five nodes. Thus, the number of nodes in the fault tree for a *leader* spacecraft of PAM multi-agent system product line (MAS-PL) was reduced by over 22% from the number of failure nodes from the original PL-SFTA. Since the product-line member considered in this case, a *leader* spacecraft of PAM MAS-PL, contained none of the variabilities of this particular fault tree, the pruned fault tree represents the common parts of the fault tree that will remain for all member of this product line. Thus, specifically for this PL-SFTA, approximately 78% of the fault tree can be reused for all 160 unique types of spacecraft for the PAM MAS-PL. Further, PLFaultCAT was able to accomplish most of the work automatically. In this case, PLFaultCAT did 72% of the pruning of a PL-SFTA possible and only required a 28% of the effort to be done manually.

Similarly, for the “A spacecraft received solar radiation damage” fault tree, discussed in Section 5.3.3.2, it was found that approximately 60% of the tree was common to all 160 unique types of spacecraft for the PAM MAS-PL and that PLFaultCAT was able to automatically do 72% of the pruning. A further discussion on the results of the application of the PL-SFTA technique described in this section and its impact on reusability and safety are provided in Section 5.3.8.

5.3.6 Accommodating Evolution in the Product-Line Software Fault Tree Analysis

It is often the case that additional variabilities are added as features to the initial product line (e.g., new scientific goals are desired in the PAM case study requiring new onboard scientific equipment and software functionality). To safely include the new

variabilities, we must perform a limited amount of domain engineering and hazard analysis to incorporate the new variabilities in order to ensure that future systems are safe. In particular, new variabilities as well as new values for existing variabilities must iterate the relevant steps in the two-phase framework illustrated in Figure 32 and Figure 33. This includes modifications to the requirements specification (as needed), as well as to the Commonality Analysis (CVA) and Software Failure Modes, Effects and Criticality Analysis (SFMECA) if they are affected.

In addition, the PL-SFTA is updated to incorporate the changes. If an SFMECA was constructed, the addition of variabilities can add new rows to the SFMECA table(s) or change the failures or effects in already existing rows in the SFMECA table(s). The PL-SFTA_CREATE algorithm, as detailed in Section 5.3.3.2, analyzes the new SFMECA rows and any additions to the preexisting SFMECA rows that can be influenced by the inclusion of the new variabilities. Following this process incorporates the new variabilities into the PL-SFTA by including their causal event nodes into the fault trees. The graphical view of the fault tree that PLFaultCAT provides makes updating the PL-SFTA to incorporate new variabilities (features) and to derive a new product-line member's SFTA efficient enough for it to be practical for projects to maintain the fault tree as a current product-line artifact.

5.3.7 An Alternative Approach for Product-Line Software Fault Tree Analysis Specific to Multi-Agent System Product Lines

In Section 5.3.3.2, a general approach was provided to construct a product-line software fault tree analysis (PL-SFTA) for a safety-critical product line. In this approach, product-line requirements were associated with the leaf nodes (see Section 5.3.3.2) of a fault tree so that product-line member's fault tree can be derived from its set of

variabilities (see Section 5.3.5). This is a rather fine-grained approach since it uses the requirements to relate a product to the nodes of a fault tree.

In the design and development of a multi-agent system product line (MAS-PL), described in Chapter 4, the product-line requirements are refined and implemented in the roles and variation points possible in an agent (see Section 4.2). Since an agent (i.e., a product-line member of the MAS-PL) is defined by the roles and the variation points possible in its roles, it may be desirable to associate the leaf nodes of a fault tree with the variation points rather than the requirements [19]. This is a coarser-grained approach since it uses the roles and variation points, typically composed of a set of requirements, to relate a product to the nodes of a fault tree.

In the previous sections, we illustrated the construction of the PL-SFTA for the Prospecting Asteroid Mission (PAM) MAS-PL by associating the requirements of the MAS-PL with the leaf nodes of a fault tree. In this section, we provide our approach to construct a PL-SFTA specifically for a MAS-PL associating the variation points to the leaf nodes of a fault tree rather than the requirements. Note that many of the steps provided here are similar to those discussed previously. Yet, to highlight the differences and provide completeness for our PL-SFTA for safety-critical product lines and MAS-PL, we briefly discuss the steps here. In [19], we have found that using the variation points with the leaf nodes of a fault tree is useful when describing a MAS.

To build a PL-SFTA for any given role and its associated variation points, the following steps should be taken:

1. Determine the root node of the fault tree. The root node is a hazard of concern in the system. It may come directly from a negation of one of the safety properties listed in the Safety Properties section of one of the Role Schema or the Variation Point Schema (see Section 4.2) or from a previously determined domain-specific hazards list.

2. Repeatedly generate a list of causes for each failure event starting at the root node. This process continues until the desired granularity is achieved. This process heavily relies on domain knowledge, previous experience and in-depth requirements analysis. The causes for each failure may come from requirements of any, all or a combination of a role's variation points. Alternatively, a Software Failure Modes, Effects and Criticality Analysis (SFMECA) can be constructed from the Variation Point Schema (see Section 5.2) and can be used to aid this step (see Section 5.2).

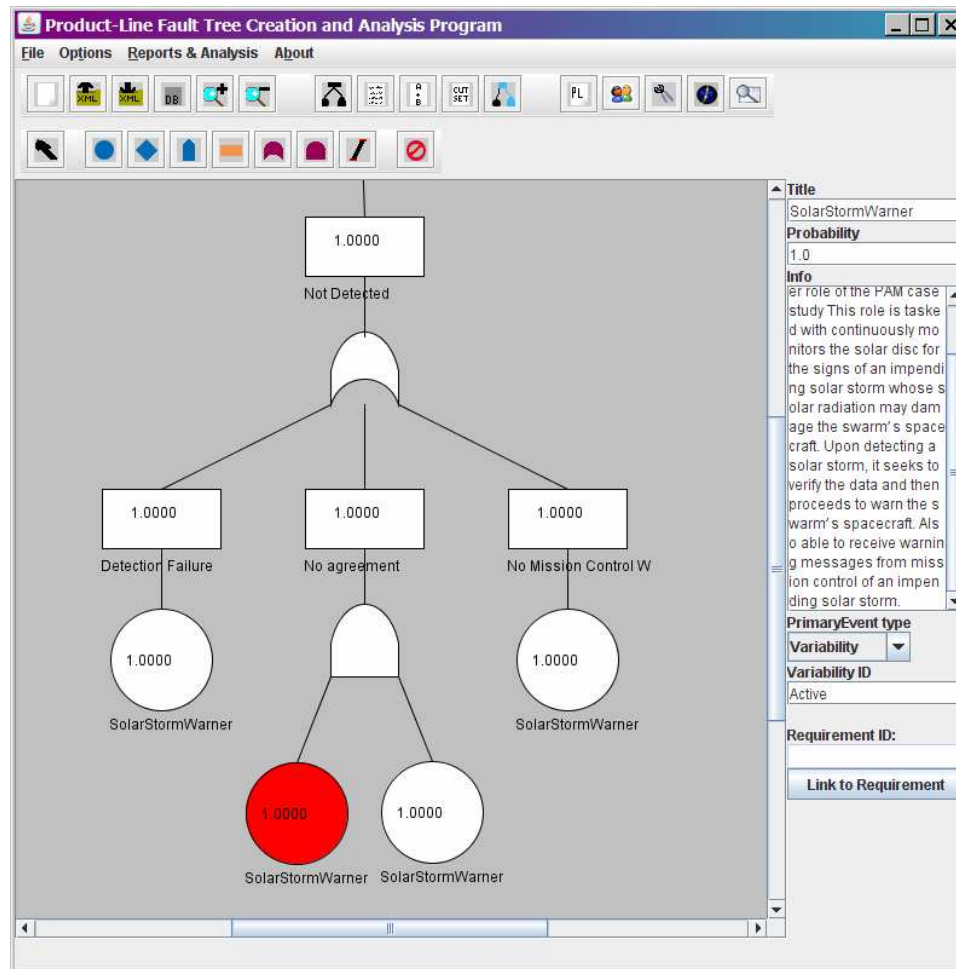


Figure 54 Depicting the Influence of a Role's Variation Point for the Spacecraft Received Solar Damage Fault Tree in PLFaultCAT

3. From the list of failures and causes generated in the previous step, construct a tree connecting the causes of a failure to a failure by logical AND or OR gates. This tree should now resemble a traditional software fault tree [44].
4. Input the constructed software fault tree to the PLFaultCAT tool.
5. For each of the leaf nodes of the resulting software fault tree in PLFaultCAT, consider which role's variation points are the source of the fault and tag the node accordingly. As shown in Figure 54, to tag the leaf node so that it is associated with a variation point, we again use a circular node in PLFaultCAT and document the name of the variation point that can cause the leaf-node failure. Note that this is the same fault tree as shown in Figure 41 but associating a variation point rather than a requirement to the fault tree's leaf node. It is possible that more than one variation point is tagged to one leaf-node failure. In this case, the tags representing a variation point should be connected to the leaf node via an OR gate (since for any given role only one variation point can be active at any given time).

These steps yield a PL-SFTA in which every leaf node is associated with one or more of the role's variation points or an external event.

After constructing and inputting a PL-SFTA of a MAS-PL using the Variation Point Schema requirements specification template, we can automatically generate the software fault tree for a particular role regardless of which variation points it contains for any given member of a MAS-PL. Using PLFaultCAT, we specify which variation points a member has, shown in Figure 55, and PLFaultCAT automatically trims the PL-SFTA to produce a fault tree specific to the combination of variation points selected just as described in Section 5.3.5.

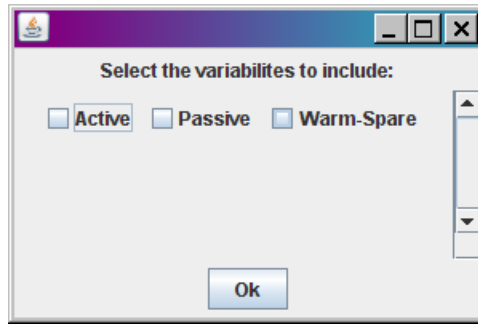


Figure 55 Variation Point Selection to Derive an Agent’s PL-SFTA in PLFaultCAT

This mechanism of PL-SFTA construction and PLFaultCAT utilization provides an initial safety analysis of an agent's role (including its variation points). This approach also allows for reuse of some of the safety analysis artifacts in that the PL-SFTA can then be automatically derived for any agent employing the same role and a combination of the role's possible variation points:

1. Determine how the new variation point can contribute to the root node hazard and each non-leaf node of the fault tree.
2. Repeatedly generate a list of causes for each new failure event created from the previous step.
3. From the new list of failures and causes, add the new nodes in PLFaultCAT to the previously constructed fault tree.
4. For each leaf node of the updated fault tree, consider whether the new variation point can contribute to the fault. Those leaf nodes that can be caused by the new variation point are tagged in a similar manner as when the fault tree was originally created.

This process produces an updated PL-SFTA within PLFaultCAT such that fault trees can be automatically generated using the new variation point and the previously documented variation points.

The use of a SFT in this manner provides software engineers some assurance that the system requirements are safe (i.e., will not contribute to the hazards). In the PAM MAS-PL case study, a PL-SFTA for the possibility of the failure "A spacecraft received solar radiation damage", also discussed in Section 5.3.3.2, for the role "SolarStormWarner", described in Sections 4.2.2.3 and 5.3.3.2 for the variation points (Passive, Warm-Spare and Active) may provide some assurance that the mission-critical system is not vulnerable to this single-point failure. Using PLFaultCAT as described in Section 5.3.2 and 5.3.4, designers can quickly generate software fault trees for all variation point combinations of the SolarStormWarner role after the initial construction of the PL-SFTA. This is both more efficient and more effective than serially constructing all the trees from scratch for the power set of the variation points (Passive, Warm-Spare and Active) for the SolarStormWarner role.

5.3.8 Evaluation and Discussion

In the domain engineering phase PLFaultCAT did not provide any significant advantages over other fault tree representation tools beyond providing the analyst with an additional opportunity to embed textual hazard analysis information into the fault tree. This allows a cross-check of the information provided in the fault tree with previously derived safety requirements, the Software Failure Modes Effects and Criticality Analysis (SFMECA) and other hazard analysis documents.

In the domain engineering phase, the application of the Product-Line Software Fault Tree Analysis (PLSFTA) to the Prospecting Asteroid Mission (PAM) case study developed four fault trees, given in Appendix F, to analyze the safety-critical hazards indicated by the requirements. This PLSFTA included 85.7% of the PAM's commonality requirements and 72.5% of its variability requirements. That is, 85.7% and 72.5% of the

Table 9 Results of the Application of PL-SFTA to the PAM Case Study

Hazard	Total Failure Nodes	Common Failure Nodes	% Commonality Requirements	Core Reuse	PLFaultCAT Automation
Spacecraft to Asteroid Collision	82	64	88.5%	78.0%	72.2%
Spacecraft to Spacecraft Collision	84	61	63.8%	72.0%	82.1%
Spacecraft Solar Storm Damage	87	52	60.0%	59.8%	72.2%
Failure to Detect Solar Storm	91	13	6.7%	14.3%	93.8%

PAM requirements, respectively, were associated to at least one of the leaf nodes in the set of fault trees of the PAM's PLSFTA.

In the application engineering phase, however, PLFaultCAT provided significant advantages from a reuse perspective by exercising the pruning method outlined in Sections 5.3.5. In the PL-SFTA considered for the PAM case study considered in this dissertation, the PL-SFTA was found to contain approximately 54% failure nodes that would be common to all 160 unique spacecraft of the PAM multi-agent system product line (MAS-PL). That is, the minimum expected reuse of the PL-SFTA for any given PAM spacecraft would be 54%. This calculation and the specific data leading to this result are described next.

Table 9 provides the results for each of the hazards examined using PL-SFTA for the PAM case study. The "Hazard" column represents the root node hazard of a fault tree in the PL-SFTA (see Appendix F, page 311); the "Total Failure Nodes" column represents the total number of failure nodes of a fault tree as build in Steps 1-3 described in Section 5.3.3.2; the "Common Failure Nodes" column represents the number of failure nodes that will be common to all product-line members of the PAM MAS-PL (i.e., pruning the PL-SFTA for all variabilities); the "% Commonality Requirements" represents the percentage of the requirements associated to the leaf nodes of a fault tree that are product-line commonality requirements; the "Core Reuse" column represents the percentage of the failure nodes that are common to all product-line members of the PAM

MAS-PL (i.e., the common failure nodes of a fault tree divided by the total number of failure nodes of a fault tree); and, finally, the “PLFaultCAT Automation” column represents the percentage of the nodes that could be safely and automatically pruned from the PL-SFTA using PLFaultCAT.

Although the overall reuse of the PL-SFTA for any spacecraft of the PAM MAS-PL developed in this dissertation is approximately 54%, in most cases the reuse potential of a fault tree in the PL-SFTA for a specific PAM spacecraft was in the 60%-80% range. The only exception was the “Failure to Detect a Solar Storm” fault tree which only had a minimum of a 14% reuse potential.

The contributing factor of a lower reuse potential of the PL-SFTA is its relation to the commonality and variability requirements. For example, the “Spacecraft to Asteroid Collision”, “Spacecraft to Spacecraft Collision” and the “Spacecraft Solar Storm Damage” fault trees (see Table 9) all had root nodes directly related to the C_SP3, C_SP1 and C_SP6, respectively product-line commonalities of the PAM MAS-PL (see Appendix A, page 234). Since each of these requirements necessitates a PAM spacecraft to prevent the hazards outlined in these fault trees, all PAM spacecraft will be equipped with the functionality to prevent the hazard. Thus, the reuse of these fault trees in the PL-SFTA is great. However, for a hazard that stems from a product-line variability, such as the “Failure to Detect Solar Storm” hazard, the reuse potential is much less since the failure of the product-line variability to mitigate against the hazard is only found in a subset of the product line’s members. Note that, however, this particular fault tree’s reuse would be significantly higher (i.e., in the 80%-100% range) for those spacecraft that have the capability to monitor the solar disc for impending solar storms (i.e., the SolarStormWarner role’s Warm-Spare or Active variation point, see Appendix C, page 298).

Further, this case study found that, of failure nodes that could be safely pruned from a PL-SFTA to derive the fault trees for a new product-line member, PLFaultCAT was able to automatically perform a minimum of 72% of the trimming without losing necessary information according to the PL-SFTA_SEARCH algorithm. Thus, 28% of the work was left to be done manually by an engineer. This metric reflects the effort saved in reuse of the PL-SFTA.

The automation that PLFaultCAT can provide when pruning a PL-SFTA for a specific member(s), is sensitive to the number of Boolean AND gates in the fault tree. As a result of the conservative pruning approach of the PL_SEARCH algorithm described in Step 2 of Section 5.3.5.1, PLFaultCAT will not automatically remove the AND gates as a safety precaution. Thus, PLFaultCAT will provide a larger amount of automated pruning for those fault trees of a PL-SFTA with fewer AND gates. Despite this, in the PAM case study we found that the automation to manual effort was at least a 3:1 ratio.

These results compare to those of a previous case study we performed in [24] on Weiss and Lai's Floating Weather Station (FWS) product line. This case study, unlike the PAM case study presented in this dissertation, was for a smaller, traditional software product line (i.e., not agent-based). In the FWS study, it was found that a smaller portion of the PL-SFTA, 45%, was common to all products of the product line. However, like the PAM case study, this case study found that PLFaultCAT was able to automatically prune 70% of the nodes that could safely be pruned.

The difference in the amount of common failure nodes in a PL-SFTA to all product-line members (i.e., 45% common in the FWS study, 54% common in the PAM study) is likely due to the type of application used in this case study. In the FWS study [24], the results reported in the FWS study reflect the application of PL-SFTA to a single fault tree for a case study consisting of fewer than 20 requirements evenly split between

commonalities and variabilities. More importantly however, is that the product-line members of the FWS did not share the same safety concerns as in the PAM study.

In the PAM study, every spacecraft had to be concerned with collisions with asteroids, collisions with other spacecraft and damage from the solar radiation present during a solar storm. These common safety concerns are thus reflected in the associated product-line commonality requirements. Thus, the PL-SFTA for the “Spacecraft to asteroid collision”, “Spacecraft to spacecraft collision” and “A spacecraft received solar radiation damage” fault trees had similar causes that could, for the most part, originate to the commonalities of the PAM MAS-PL. As a result, a large portion of the PL-SFTA could be reused regardless of the specific configuration of the spacecraft.

The agent characteristics of the PAM case study as well as the types of variabilities that were present in the case study had a large impact on this result. The spacecraft of the PAM case study had the inherent onus to be responsible for protecting and healing itself from the possible dangers of space exploration. For this reason, each spacecraft is to be equipped with the behavior to protect itself from the types of hazards modeled in the PL-SFTA. Further, the variabilities of the PAM MAS-PL concerned the differing types of scientific exploration possible in the spacecraft and had only a minor impact on the leaf nodes of the PL-SFTA.

The implication of this result is that a PL-SFTA may best suit a MAS-PL compared to a traditional product line since the agents of a MAS-PL will typically also include self-protecting and self-healing characteristics as commonalities and may have variabilities that are less likely to be safety-critical. However, for those traditional product lines that have few variabilities that will impact the safety of a system, a PL-SFTA can be applied and achieve the results found in the PAM study. Yet, even for those traditional product lines that may have a large number of variabilities that will impact the safety of a system, such as the FWS case study, the reusable part of the PL-SFTA is

modest and likely advantageous compared to the alternative of individually constructing the safety analysis for each different member.

A concern for performing safety analysis on safety-critical product lines is whether the technique is scalable as the product line grows more complex by incorporating more variabilities and product-line members. From the experience of applying the PL-SFTA to the PAM case study in this dissertation, it appears that our method and tool will scale adequately as the product line grows more complex. This is because most of the added complexity in a large product line lies in the domain engineering phase when the PL-SFTA is constructed. In Chapter 4 we provided a structured process to construct the SFMECA for a MAS-PL from the Variation Point Schemas using the Gaia-PL methodology. Since, the construction of the PL-SFTA described in Section 5.3.2 relies heavily on the aid of a SFMECA, the scalability is at least as robust as that of the SFMECA. Additionally, it should be clear that the reuse of the product-line fault tree approach is far more efficient especially for large product lines than to serially construct SFTAs for each of the desired product-line members of a product line.

The communicability of a PL-SFTA created in PLFaultCAT with other applications is high since PLFaultCAT provides a user with three different views of any given fault tree: a standard graphical fault tree view, an XML file view and a text-based view. This variety of PL-SFTA views should allow PLFaultCAT's integration into other safety analysis techniques and tools. The XML output file utilized in PLFaultCAT supports straightforward linking with existing static analysis tools. For example, the use of a PL-SFTA created in PLFaultCAT with other applications (such as Relex or Galileo) would at most only require a translation program to mediate the format of the XML file.

Finally, it should be noted that the reliance on domain expertise and knowledge of the proposed system to construct the PL-SFTA only guarantees that the PL-SFTA is only

as good as the engineer creating it. Thus, although this chapter provides a set of structured steps to guide the construction of a PL-SFTA using a SFMECA, the responsibility of the accuracy and completeness of the fault trees of a PL-SFTA lies on the software engineers rather than the PL-SFTA. However, Section 5.4 describes how the SFMECA and PL-SFTA can be used in a Bi-Directional Safety Analysis (BDSA) to aid in the completeness checking of the PL-SFTA.

5.3.9 Using the Product-Line Software Fault Tree Analysis to Aid Other Safety Analysis Techniques

In addition to the safety analysis opportunities that the product-line Software Fault Tree Analysis (PL-SFTA) offers in PLFaultCAT, discussed in Section 5.3.4, PL-SFTA and PLFaultCAT can be used to support and guide other safety analysis techniques for safety-critical product lines. In particular, PL-SFTA and PLFaultCAT have been shown to be useful in providing guidance for the safety analysis for software product lines using a state-based modeling approach [45], [46], [47], [48]. This approach provides software engineers with a structured way to build state-based models for a safety-critical product line, systematically explores the relationships between the software's behavioral variations and potential hazardous states and supports the automated verification of safety properties across a product line.

To support the state-based modeling approach, a PL-SFTA was used to derive the required scenarios (i.e., those scenarios that enforce a safety property) and forbidden scenarios (i.e., those scenarios that emulate a hazard) to exercise against a state model. In addition, PLFaultCAT aids in identifying the safety-critical feature interactions by searching for those product-line requirements that frequently contribute to the possible causes of the fault tree's failure nodes. PLFaultCAT can automatically identify those product-line requirements and combination of product-line variabilities (i.e., features)

that contribute to the most potential failures as defined in the PL-SFTA, as was described in Section 5.3.4.

This analysis provides a prioritized list of those product-line requirements and feature interactions that warrant further scrutiny using an executable state-based model. That is, those product-line requirements and feature interactions that are deemed to contribute to the most fault tree failure nodes are more likely to have unsafe interactions with existing product-line requirements and should have their behaviors modeled in order to determine the safe/unsafe behaviors using a dynamic analysis.

The use of a state-based modeling approach for safety analysis is advantageous because it can both build on and extend the PL-SFTA. Unlike a PL-SFTA, an executable state-based model can analyze and model the timing/ordering of failure events to determine their possible safety implications. In addition, we found that because the SFTA is a static asset, it lacks the ability to animate and explicitly show how a safety property may be violated [45], [48]. The use of an executable state-based model, however, allows the simulation of the behaviors described by the requirements in the fault tree to illustrate the violation of a safety property.

Moreover, the executable state-based model, unlike the PL-SFTA, can explore multiple solutions to come up with a reliable and easy-to-implement mitigation strategy. This then drives the updating of the product line's requirements to include the new safety requirements. Such feedback is impossible to ensure using the PL-SFTA alone. Thus, the inclusion of a state-based modeling safety analysis approach may improve the safety case that a safety-critical product line must make when requiring certification from an outside governing body.

The use of the PL-SFTA technique and its tool, PLFaultCAT, in concert with the state-based provides software engineers with a set of tools to best assess the safety of a software system and make it more practical for software engineers to check the behavior

of product variations for potential safety consequences as well as enhancing the models reusability as a safety asset for new products. Note that this line of research is not the primary work of this author and thus is not the focus of this dissertation. A full description of the safety analysis for software product lines using a state-based modeling approach is provided in [45], [46], [47], [48].

Although we have only demonstrated the state-based modeling safety analysis techniques on a traditional product line (i.e., not an agent-based system), their application towards a multi-agent system product line (MAS-PL) should be straightforward and would further provide safety analysis techniques that can both analyze a MAS-PL and provide reusable safety analysis assets for future systems. Section 6.2 further details this approach as Future Work.

5.3.10 Summary

This section detailed and illustrated our extension of the traditional Software Fault Tree Analysis (SFTA) technique to an entire product line with the support of a software tool, PLFaultCAT on a safety-critical multi-agent system product line (MAS-PL). This extension supports the construction of a product-line SFTA (PL-SFTA) in PLFaultCAT from common hazard analysis assets during the domain engineering phase of software product-line engineering. We showed how new safety requirements can be discovered and mitigations to possible hazards can be introduced through the introduction of new product-line requirements or constraints. This section also presented the pruning technique developed and implemented in PLFaultCAT during the application engineering phase to derive the SFTA for single product members of the product line.

The Software Failure Modes, Effects and Criticality Analysis (SFMECA), described in Section 5.2, and the Product-Line Software Fault Tree Analysis (PL-SFTA), described in this section, can be viewed as complementary since the output of the

SFMECA (i.e., the potential system-wide hazards) should match-up with the inputs of the PL-SFTA. Similarly, the output of the PL-SFTA (i.e., the low-level, local errors that cause a system-wide hazard) should match-up with the inputs of the SFMECA. The comparison of a SFMECA, a forward analysis technique, and a PL-SFTA, a backward analysis technique, is used in a Bi-Directional Safety Analysis to help ensure consistency and completeness. The next section describes the BDSA for a MAS-PL using the SFMECA developed in Section 5.2 and the PL-SFTA developed in Section 5.3.

5.4 Bi-Directional Safety Analysis for Multi-Agent System Product Lines

The development of a forward and backward safety analysis technique for a safety-critical, multi-agent system product line (MAS-PL) was partly motivated by the opportunity to perform a Bi-Directional Safety Analysis (BDSA) on the design of a MAS-PL to better provide assurance of its safety. The results of a forward search, such as the Software Failure Modes Effects and Criticality Analysis (SFMECA) described in Section 5.2, and a backward search, such as a product-line Software Fault Tree Analysis (PL-SFTA), will not necessarily be the same, often times both types are utilized in the safety analysis of a safety-critical system [44].

The SFMECA and PL-SFTA techniques developed in this work can be viewed as complementary since the output of the SFMECA (i.e., the potential system-wide hazards) should match-up with the inputs (i.e., high-level or root nodes) of the PL-SFTA. Indeed, in Section 5.3.3.1 it was mentioned that one source of the hazards to model as a root node of a PL-SFTA can be the SFMECA tables. Similarly, the output of the PL-SFTA (i.e., the low-level, leaf node failures of a fault tree) should match-up with the inputs (i.e., local effects column) of the SFMECA. For example, we can verify the completeness of the SFTA by ensuring that every unique hazard listed in the SFMECA table with a particular

level of criticality or higher (e.g., major criticality) is a root node within one of the fault trees of the SFTA. Thus, BDSA helps to ensure that the safety analyses used for the forward and backward search techniques are consistent for a safety-critical software product line.

This section details a structured process to perform a BDSA tailored to the requirements specification of a safety-critical MAS-PL generated from the Gaia-PL methodology (see Chapter 4) using the SFMECA (see Section 5.2) and PL-SFTA (see Section 5.3).

5.4.1 Assessing Gaia-PL's Requirements Specifications using Bi-Directional Safety Analysis

To assess and derive safety requirements of the Role Schemas and the Variation Schemas from Gaia-PL using the SFMECA, the following steps suffice:

1. For each Role Schema and Variation Point Schema:
 - a. For each data/event listed in the *Data/Event* column of the SFMECA for the role represented in the Role Schema / Variation Point Schema:
 - i. Decide at which level of criticality (i.e., critical, major, etc.) the role must provide mitigating requirements to ensure safety. This may correspond to what level of system certification is required of the system.
 - ii. For each listed data/event failure mode listed in the *Failure Mode* column of the SFMECA with a criticality of at least the minimum criticality level needed for analysis (from Step i):
 - a. Consult the local effect of the failure mode in the *Local Effect(s)* column of the SFMECA. Assure that the software mitigates the local effect. For data, the mitigating

requirement could be some sanity check (i.e., checking some other piece of data or monitoring that the data is reasonable given the software’s current state). For events, the mitigation requirement could be some guard to ensure that the event is occurring under the right conditions and at the appropriate time given the software’s current condition.

- b. Check to make sure that the MAS-PL software will prevent the hazard described in the *Possible Hazard* column of the SFMECA from occurring in the PL-SFTA. That is, check that the hazard is mitigated in **both** the SFMECA and PL-SFTA.
- c. If the mitigation does not suffice to prevent the local effect, the software may not be compliant with system safety requirements.

For example, applying this process to the Prospecting Asteroid Mission (PAM) case study used in this dissertation identified several new mitigation requirements to prevent the hazard of a “spacecraft to asteroid collision” that were then added to the Variation Point Schema. For the “halt/abnormal termination” failure mode for the SFMECA given in Table 5, the mitigation requirement was that the *MoveToAvoidCollision* activity be atomic (either it executes completely or not at all). Alternatively, a new NotifyFinishMoveNewPos protocol could be introduced to have the spacecraft notify nearby spacecraft (or the *leader* spacecraft in charge of the subswarm) of the completion (or non-completion) of the *MoveToAvoidCollision* activity. Additionally, a mitigation requirement for the “timing/order” failure mode could be to assign a timestamp deadline by which each *MoveToAvoidCollision* activity must

complete before. Without the BDSA and SFMECA process detailed above, safety requirements such as these could be overlooked.

5.4.2 Bi-Directional Safety Analysis's Role in Strengthening the Safety Case of a Multi-Agent System Product Line

For the multi-agent system product line (MAS-PL) applications of the future, safety certification may be desired or required before the system can be deployed. Certification is a process whereby a certification authority determines if an applicant provides sufficient evidence concerning the means of production of a candidate product and the characteristics of the candidate product so that the requirements of the certifying authority are fulfilled [31], [40], [69], [72]. Certification may apply to the development process, the developer or the actual product [55]. Since it is insufficient to certify the process or developer for the software of safety-critical systems, building a safety case that provides “an argument accompanied by evidence that all safety concerns and risks have been correctly identified and mitigated” [26] aids in the certification of the product.

The safety analysis techniques and tools described in this chapter integrate the reuse potential of safety analysis assets into the design and development of a safety-critical MAS-PL so that they can be used to better make a safety case when system certification is required as well as allowing the safety engineer to verify the safety requirements of the system and can discover missing safety requirements. These safety analyses provide some assurance that core assets defined in the domain engineering phase are being safely reused during the application engineering phase.

In addition to strengthening the safety case of a MAS-PL using the process described in Section 5.4.1, the BDSA can contribute to the certification of a safety-critical MAS-PL. Specifically, the use of BDSA can assist in certification of MAS-PL in two ways:

- *Demonstration of compliance.* The use of BDSA provides assurances that certain classes of failure modes that might occur in the individual agents will not produce unacceptable effects in the composite system (e.g., the constellation, or fleet). The artifacts produced in this investigation (SFMECA tables, PL-SFTAs, and the Role Schemas and Variation Point Schemas responsibility statements) help demonstrate compliance of the failure-monitoring and failure-mitigation software tasked with the system safety requirements.
- *Enabling reuse of certification arguments.* The use of product BDSA can reduce the burden of certification for systems composed of identical or near-identical units (e.g., the Prospecting Asteroid Mission (PAM) case study used in this dissertation). In systems where each agent is a member of a product line, the similarities can be leveraged for efficient reuse of the safety analysis artifacts. At the same time, the use of Role Schemas and Variation Point Schemas captures any variations among the roles that the agents may assume. The Role Schemas and Variation Point Schemas thus help ensure that the reuse of the artifacts in the certification arguments accurately reflects any differences among the agents.

Thus, the use of BDSA can greatly improve the effectiveness of the safety analysis artifacts of a safety-critical MAS-PL.

5.5 Summary

This chapter detailed our safety analysis techniques and tools for the analysis of safety-critical multi-agent system product lines (MAS-PL). We detailed three safety analysis techniques: product-line Software Fault Tree Analysis (PL-SFTA), Software

Failure Modes, Effects and Criticality Analysis (SFMECA) and Bi-Directional Safety-Analysis (BDSA).

PL-SFTA and its tool, PLFaultCAT, provide the capability to construct a software fault tree for a product line and then reuse the PL-SFTA to automatically derive the fault trees for individual product-line members. We detailed how to build a product-line fault tree by associating the leaf nodes of a fault tree to the related product-line requirements. For a PL-SFTA, we showed how PLFaultCAT can automatically analyze the set of PL-SFTAs for single-point failures and automatically identify safety-critical requirements and requirement interactions using PLFaultCAT.

The SFMECA safety analysis technique was incorporated into our Gaia-PL methodology to produce a safety analysis technique specifically for a safety-critical MAS-PL. We provided a structured process to analyze the Variation Point Schemas produced in the Gaia-PL methodology to discover the ways in which the agents of the MAS-PL can fail and the effects of the failures on the entire system. The information generated here can aid in discovering missing safety requirements, designing mitigation requirements to prevent failures and verify existing safety requirements.

Finally, we detailed how the SFMECA derived from the Gaia-PL assets and the PL-SFTA can be used together to perform a BDSA on the safety analysis assets of a MAS-PL improves the SFMECA and PL-SFTA by identifying incompleteness in both safety analyses. This aids in strengthening the safety analyses of the MAS-PL and provides further opportunities to discover missing safety requirements. Further, the BDSA process described additionally contributes to system certification by verifying software design compliance with reliability, robustness and safety standards by strengthening the safety case when the demonstration of the compliance of failure-monitoring and failure mitigation software tasked with the safety requirements to safety standards in the MAS-PL is necessary.

This chapter's objective was to be able to provide safety analysis artifacts for a new system in MAS-PL in a timely, cost-effective and safe manner. The safety analysis techniques and tools presented in this chapter should provide software engineers with a set of instruments to help build safety-critical MAS-PL in such a way that the safety analyses assets can be reused for future systems.

CHAPTER 6. CONCLUSION

Chapter 1 presented the following thesis statements for the work presented in this dissertation: *an AOSE methodology can be devised to enhance the reuse in the design and development of a safety-critical MAS by incorporating software product-line engineering principles to develop reusable software engineering assets in a way that allows software engineers to take advantage of the reusable assets to create MAS; and that product-line safety analysis techniques and tools can be developed and adopted to support the development of a safety-critical MAS by discovering, analyzing and verifying the MAS's requirements in a way that produces reusable safety assets that can be used for future systems of the MAS.*

This chapter concludes this dissertation with a discussion of how this work supports the two claims of the thesis and a summary of the contributions. Future avenues of research stemming from this dissertation are presented and, finally, concluding remarks are provided reflecting on the motivation, contributions and application of this research.

6.1 Support for the Thesis

Chapter 2 first presented the background information and related research that lay the foundation for the AOSE methodology, Gaia-PL (Gaia – Product Line), presented here. Software product-line engineering is an established approach to reusing software development assets as a mechanism to reduce the development cost of software systems developed within a software product line. AOSE is an emerging software engineering field to design and develop highly distributed, intelligent software systems. Gaia-PL introduces and incorporates ideas from software product-line engineering into AOSE so that agent-based systems can take advantage of the reuse inherent in software product-line engineering to achieve a reduction in development cost. This chapter also discussed

the related approaches in these areas to identify the differences of previous work from the Gaia-PL approach described in Chapter 3.

Chapter 2 also discussed software safety analysis techniques and tools in the context of software product lines. Software safety analysis techniques, including Software Fault Tree Analysis (SFTA), Software Failure Modes, Effects and Criticality Analysis (SFMECA) and Bi-Directional Safety Analysis (BDSA), provide the groundwork for the product-line SFTA (PL-SFTA) technique, and its associated tool, PLFaultCAT, developed in this work. In addition, these safety analysis techniques were specifically adapted and included into our Gaia-PL AOSE methodology to aid in the safety analysis of MAS product lines (MAS-PL) in a way that is partially reusable.

Next, Chapter 4 detailed the design and development of a MAS-PL using our Gaia-PL AOSE methodology on the PAM case study. The Gaia-PL methodology produces reusable software engineering assets so that building systems of the MAS-PL can be done efficiently, in terms of development cost and time. First, we described how we adopted software product-line engineering concepts into AOSE by identifying, defining and using *variation points* to build MAS. We then illustrated the adaptation of software product-line engineering's Domain Engineering phase into Gaia's Requirement Documentation and Analysis and Design phases. In these phases, we illustrated the documentation of MAS-PL requirements in a Commonality and Variability Analysis and a Parameters of Variation table.

We detailed the documentation of requirement specifications in the Role and Role Variation Point Schemas. These schemas partitioned the commonality requirements and variability requirements into separate schemas for specific roles using a Feature Model as a guide. We described the adaptation of software product-line engineering's Application Engineering phase into Gaia's Detailed Design phase. In this phase, we illustrated the

reuse of the Role and Role Variation Point Schemas to build specific types of agents for a MAS-PL.

We then discussed and illustrated the reuse of the requirements specifications during initial system development of a MAS-PL as well as during system evolution. To highlight the advantages of Gaia-PL, we differentiated our methodology from previous work by illustrating Gaia-PL's ability to capture reuse and avoid the redundant work and increased development cost (in the additional time required) necessitated to accommodate the development of the agents as done in previous work. Finally, Chapter 4 concluded by providing an evaluation of our Gaia-PL methodology on the PAM case study to illustrate the reusability, development cost savings and other advantages of our approach.

Chapter 5 discussed our safety analysis techniques and tools for the analysis of safety-critical software product lines and MAS-PL. First, we discussed our adaptation of Software Failure Modes, Effects and Criticality Analysis (SFMECA) in our Gaia-PL AOSE methodology to produce a safety analysis technique specifically for safety-critical MAS-PL. We provided a structured process to analyze the Variation Point Schemas produced in the Gaia-PL methodology to discover the ways in which the agents of the MAS-PL can fail and the effects of the failures on the entire system. The information generated here can aid in discovering missing safety requirements, designing mitigation requirements to prevent failures and verify existing safety requirements.

Next, our product-line Software Fault Tree Analysis technique (PL-SFTA) and its tool, PLFaultCAT were discussed. After detailing PLFaultCAT's software architecture, the construction of a PL-SFTA was discussed during software product-line engineering's Domain Engineering phase using PLFaultCAT. We illustrated how to build a product-line fault tree, link the leaf nodes to a product-line requirement documented in DECIMAL, automatically analyze the set of PL-SFTAs for single-point failures and

automatically identify safety-critical requirements and requirement interactions using PLFaultCAT.

For software product-line engineering's Application Engineering phase, we detailed the partially-automated pruning of the set of product-line fault trees to produce the set of fault trees for a member of a product line. PLFaultCAT takes the product-line requirements of a product documented in DECIMAL to automatically prune the branches of a PL-SFTA that can be safely removed for that specific product. That is, the branches of the PL-SFTA that are not relevant to a product because they involve requirements (i.e., variabilities) or values of variabilities that are not present or could not possibly present in the product are pruned from the fault tree. Thus, reuse in the safety analysis is achieved by reusing the PL-SFTA developed in the Domain Engineering phase for the derivation of SFTAs for the product-line members created during the Application Engineering phase.

PLFaultCAT takes a conservative approach to the pruning by only pruning those nodes of a PL-SFTA that can be safely removed. Because of this, additional manual pruning of PL-SFTA nodes may be needed to be performed by a safety engineer to derive the product-line member's SFTA. Despite this, our case study has shown that PLFaultCAT can automatically prune about 70% of the PL-SFTA nodes that can be safely removed for any given member of a product line.

It was also shown how our PL-SFTA approach can accommodate evolution of the software product line. We illustrated the process to handle the addition of product-line requirements in the PL-SFTA.

An evaluation of our PL-SFTA technique and the PLFaultCAT tool was also provided using the PAM case study to illustrate PL-SFTA's value as a reusable asset and PLFaultCAT's ability to automatically derive the SFTAs for a product-line member. This evaluation shows how a PL-SFTA can capture the common parts of a SFTA and reuse

them for the members of a product line avoiding the cost that would be incurred if producing the same products serially (i.e., using the traditional SFTA approach rather than our PL-SFTA approach). The application of PL-SFTA to the case study used throughout this dissertation illustrated that an average of 54% of the PL-SFTA can be reused for the product-line members. Further, we showed PLFaultCAT's capability to increase the safety of a product line by identifying new and missing safety requirements by utilizing PLFaultCAT's novel features to analyze the PL-SFTA.

Next, we detailed how the SFMECA derived from the Gaia-PL assets and PL-SFTA can be used together to perform a Bi-Directional Safety Analysis. Performing a BDSA on the safety analysis assets of a MAS-PL improves the SFMECA and PL-SFTA by identifying incompleteness in both safety analyses. This improves the safety analyses of the MAS-PL and provides further opportunities to discover missing safety requirements.

The BDSA process described additionally contributes to system certification by verifying software design compliance with reliability, robustness and safety standards. The application of BDSA to a MAS-PL can assist in the certification by providing assurances that classes of failure modes that could occur in individual agents will not produce unacceptable effects in the entire MAS. This aids in strengthening the safety case when the demonstration of the compliance of failure-monitoring and failure mitigation software tasked with the safety requirements to safety standards in the MAS-PL is necessary. Further, the BSDA process described in Chapter 5 was shown to enable the reuse of safety certification arguments while ensuring that the reuse of the safety analysis artifacts in the certification argument accurately reflect the differences amongst the agents of the MAS-PL.

6.2 Summary of Contributions

This dissertation makes contributions in three key areas. First, the Gaia-PL methodology provides Agent-Oriented Software Engineering (AOSE) with a design and development methodology for agent-based systems that can reduce the development cost by taking advantage of the reuse principles of software product-line engineering. Second, the product-line Software Fault Tree Analysis (PL-SFTA) technique and its tool PLFaultCAT provide software engineers developing a safety-critical product line with a tool-supported technique to create a PL-SFTA and automatically derive the product-line members' SFTA. Third, the integration of Software Failure Modes, Effects and Criticality Analysis (SFMECA) and Bi-Directional Safety Analysis (BDSA) into Gaia-PL, along with the demonstration of PL-SFTA for MAS-PL, aids in system certification and the discovery, analysis and verification of a MAS-PL's safety requirements.

The Gaia-PL methodology was initially described at the *2005 International Conference on Software Engineering's Workshop on Software Engineering for Large-Scale, Multi-Agent Systems* [19]. It was expanded in a 2006 edition of *Lecture Notes in Computer Science* [21] as well as in a chapter in a forthcoming book entitled *Agent-Oriented Software Engineering* [62].

This dissertation has further expanded the Gaia-PL methodology from the previously published work by including a Feature Model to aid in the identification of variation points of a role, expanded its applicability to MAS-PL by introducing a more hierarchical approach and by fully evaluating the approach for its ability to decrease development cost through reuse. The specific contributions of Gaia-PL include:

- The inclusion of software product-line engineering principles into the development of MAS to build MAS-PL

- An AOSE methodology that supports the design and development of MAS-PL using aspects of Gaia, an established AOSE methodology, and FAST, an established software product-line engineering methodology
- The illustration of how Gaia-PL is amenable to the development of reusable software engineering assets during the design and development of MAS-PL and how the reusable assets can be used to develop systems of the MAS-PL
- An evaluation of Gaia-PL methodology's ability to reduce the development cost of MAS via a case study and comparison to the Gaia methodology

Our PL-SFTA safety analysis technique and the PLFaultCAT tool were initially described at the *2004 High Assurance Systems Engineering Conference* [17]. A short paper appeared at the *2005 International Symposium on Software Reliability Engineering* [18] and additional papers at the *2005 International Conference on Software Engineering's Workshop on Software Engineering for Large-Scale, Multi-Agent Systems* [19], at the *2006 Workshop on Innovative Techniques for Certification of Embedded Systems* [22], in a 2006 article in the *Automated Software Engineering Journal* [24] and in a research demonstration at the *2007 International Conference on Software Engineering* [23].

This dissertation has further extended this work to the application of a safety-critical MAS-PL and extensively evaluated the technique and tool using the PAM case study.

The specific contributions of PL-SFTA software safety analysis technique and the PLFaultCAT tool include:

- Develops fault trees for a software product line in a way that the resulting PL-SFTA is reusable for the products in a product line
- Aids in discovering additional system safety requirements for a product line
- Helps in identifying additional product-line dependencies

- Allows for an analyses to assess failure points and safety-critical requirements of a software product line
- Complements SFMECA, BDSA and other safety analysis techniques to strengthen a safety case when system certification is required
- Automatically derives all of the product line member SFTAs from PL-SFTAs
- Links product-line requirements to PL-SFTA nodes to aid in traceability
- Searches the set of PL-SFTAs to identify single-point failures
- Identifies safety-critical requirements of the entire product line by analyzing the set of PL-SFTAs
- Provides a minimum-cut set analysis of a PL-SFTA to identify hazard paths

In addition, this technique and tool have been used in collaboration with Jing Liu and Robyn Lutz as guidance for another product-line safety analysis technique that appeared at the *2005 International Symposium on Software Reliability Engineering* [47], at the *2007 Workshop on Model-Based Development* [48] and in a forthcoming article in the *Journal of Systems and Software* [45].

The inclusion of safety analysis techniques (i.e., PL-SFTA, SFMECA and BDSA) into the Gaia-PL methodology to perform safety analysis on MAS-PL was initially reported at the *2005 International Conference on Software Engineering's Workshop on Software Engineering for Large-Scale, Multi-Agent Systems* [19], in a short paper at the *2005 International Symposium on Software Reliability Engineering* [18], at the *2006 Workshop on Innovative Techniques for Certification of Embedded Systems* [22] and in a chapter in a forthcoming book tentatively entitled *Agent-Oriented Software Engineering* [62].

This dissertation has further extended this work to the application of a safety-critical MAS-PL and extensively evaluated these techniques using the PAM case study.

The specific contributions of the inclusion of safety analysis techniques into the Gaia-PL methodology for designing and developing safety-critical MAS-PL include:

- Extending BDSA to MAS-PL and showing how the analysis artifacts contribute to the software's safety case for certification purposes
- Supplying a structured process to perform SFMECA in the Gaia-PL methodology
- Providing assurances that certain classes of failure modes that might occur in individual agents will not produce unacceptable effects in the composite system, demonstrating the compliance of failure-monitoring and failure mitigation software tasked with the system safety requirements to safety standards
- Enabling reuse of certification arguments while ensuring that the reuse of the safety analysis artifacts in the certification arguments accurately reflect the differences amongst the agents of the system

6.3 Future Work

There are several avenues for future research and development based on the work and results of this dissertation, some of which involve expanding the less detailed/unexplored portions of our AOSE methodology that integrates software product-line engineering concepts, Gaia-PL. These avenues of research include (but are not limited to) the following:

- Expansion and application of Gaia-PL into the other parts of Gaia to cover a broader selection of the models and phases in the development of multi-agent system product lines (MAS-PL)
- Comparison and evaluation of our contributions to aid in the certification of agent-based software systems of our approach to others' work
- Inclusion of additional product-line safety analysis techniques into the design and development of MAS-PL

- Integration of reliability engineering techniques into our safety analysis techniques to provide reliability assurances to MAS-PL
- Adaptation of our safety analysis techniques to analyzing and verifying the security properties of a MAS-PL
- Investigation of the Gaia-PL methodology to the design and development of sensor nodes in a sensor network

The Gaia-PL AOSE methodology described in this dissertation primarily focused on the documentation and reuse of requirement specifications for a MAS-PL. This work made the initial strides into integrating software product-line engineering concepts into the design and development of agent-based software systems. To achieve this, we solely concentrated on portions of the Gaia methodology and the inclusion of reuse principles into some of its models. Thus, our Gaia-PL methodology chiefly focuses on capturing the commonalities of agents in a MAS-PL rather than providing a full suite of models and abstraction mechanisms for all phases in the design and development of a MAS-PL. Although the Gaia-PL methodology can seamlessly be integrated as a part of the Gaia methodology (i.e., using Gaia-PL's Role and Role Variation Point Schemas for the requirements and the remaining Gaia models to design and develop other parts of the MAS-PL), further work can be done to adopt other models of Gaia into Gaia-PL by further including the product line ideas discussed in this dissertation.

Alternatively, the Gaia-PL methodology may better benefit from working with other MAS-PL AOSE methodologies that have followed our work in [19], [21]. The MaCMAS AOSE methodology for designing and developing MAS-PL uses UML to model a MAS-PL and focuses on handling the complexity of MAS-PL and building its core architecture [62], [64], [65]. Thus, the use of Gaia-PL for the requirements and early design phases along with the use of MaCMAS to derive the MAS-PL's core architecture may be a natural and advantageous approach.

The initial results from an application of our safety analysis techniques (i.e., product-line Software Fault Tree Analysis (PL-SFTA), Software Failure Modes, Effects and Criticality Analysis (SFMCEA) and Bi-Directional Safety Analysis (BDSA), to the PAM MAS-PL case study, described in Chapter 5, suggests that these technologies can reduce the effort involved in certifying the safety of new systems within a MAS-PL. Yet, further investigation into the ways in which software certification can be reduced through the use of reusable safety analysis assets may be warranted. An empirical study into this as well as a comparison to similar approaches, if they exist, would benefit the AOSE community.

Other product line safety analysis techniques [45], [47], [48] developed by this author in collaboration with Jing Liu and Robyn Lutz have been shown to be effective in constructing the behavioral model of a product line's safety-critical variability requirements in order to support the automated verification of safety properties across a product line. Although we have only demonstrated these techniques on a cardiac pacemaker product line (i.e., not an agent-based system), their application towards a MAS-PL should be straightforward and would further provide AOSE with the safety analysis techniques that can both analyze a MAS-PL and provide reusable safety analysis assets for future systems.

Safety analysis and reliability engineering are both facets of software dependability engineering. Other approaches, such as Galileo [30], [60], [78], directly integrate reliability data (e.g., failure probability rates) into safety analysis techniques. The certification of some systems (e.g., aircraft, pacemakers, etc.) frequently requires calculated failure rates (i.e., 10^{-9} probability of failure for aircraft). The inclusion of reliability engineering techniques and models into the safety analysis techniques and tools described in this dissertation would further strengthen the safety case needed for the certification of a MAS-PL. However, the challenge in this would be to enhance the

autonomous (e.g., unpredictable) nature of an agent with the predictability needed by many reliability engineering techniques.

In some cases, the safety requirements and properties of interest to this dissertation have similarities to the type of security properties that would be of interest to the designers and developers of a MAS-PL. The exploration into how the safety analysis techniques developed in this dissertation, as well as other techniques, can contribute to the validation of MAS-PL's security properties as well as derive reusable assets for verifying future product line members' security properties is a natural extension of this work.

Like agent-based systems, sensor networks typically consist of similar nodes that could benefit from reuse and safety analysis mechanisms in their design and development phases. The investigation and application of the ideas developed in this dissertation for the design and development of agent-based systems may apply to the design and development of sensor networks. This avenue of research may be of great interest to the sensor network community as it would further bring the possibility of reuse, in both hardware and software, into the design and development of the nodes of a sensor node product line in order to reduce their development cost.

6.3 Summary

This dissertation offered our AOSE methodology, Gaia-PL (Gaia – Product Line) for the design and development of agent-based, distributed software systems. Gaia-PL captures requirements specifications by using a product-line perspective to promote reuse in agent-based, software systems early in the development lifecycle. This allows software engineers to be able to reuse some software engineering assets during the initial system development as well as during system evolution.

For safety-critical agent-based systems, this dissertation developed and incorporated reuse-oriented safety analysis methods for the Gaia-PL methodology to allow the discovery of new safety requirements and the verification that the design satisfies the safety requirements. Specifically, Product-Line Software Fault Tree Analysis (PL-SFTA) and its automated tool, PLFaultCAT (**P**roduct-**L**ine **F**ault Tree **C**reation and **A**nalysis **T**ool) have been created to provide the technique and tool support for the safety analysis of safety-critical software product lines and allow for the identification of new safety requirements and the analysis of safety-critical requirements and requirement interactions. An AOSE-adapted Software Failure Modes, Effects and Criticality Analysis (SFMECA) technique was created to support the derivation of a safety analysis asset using the specifications of Gaia-PL allowing for the identification of possible hazard scenarios and the failure points of specific agent roles. Using the assets generated via PL-SFTA and SFMECA, Bi-Directional Safety Analysis (BDSA) is shown to aid in the completeness of PL-SFTA and SFMECA, help verify the safety properties and strengthen the safety case when compliance to safety standards of the multi-agent system is necessary.

The goal of this work was to be able to provide safety verification results for a new system in the product line in a timely, cost-effective and safe manner. It is hoped that the contributions of the work presented in this dissertation provide software engineers with an AOSE methodology to build safety-critical, agent-based systems so that the safety analysis assets as well as the requirements analysis and design can be reused for future systems.

BIBLIOGRAPHY

- [1] Ardis, M.A. and Weiss, D.M. 1997. Defining Families: The Commonality Analysis. Proceedings 19th International Conference on Software Engineering, Boston, MA, pp. 649-650.
- [2] Arkusinski, A. 2005. A Method to Increase the Design Assurance Level of Software by Means of FMEA. Proceedings 24th Digital Avionics Systems Conference, 2:10.D.5-1-10.D.5-11.
- [3] Bresciani, P., Giorgini, P., Guinchiglia, F. and Perini, A. 2004 TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems, 8(1):203-236.
- [4] Burgess, M. 2003. Fault Tree Creation and Analysis Tool: User Manual. <http://www.iu.hio.no/FaultCat> (Accessed May 2007).
- [5] Castro, J., Kolp, M. and Myopoulos, J. 2002. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems 27(6):365-389.
- [6] Cernuzzi, L., Juan, T., Sterling, L. and Zambonelli, F. 2004. The Gaia Methodology: Basic Concepts and Extensions. Methodologies and Software Engineering for Agent Systems-The Agent-Oriented Software Engineering Handbook Series: Multiagent Systems, Artificial Societies, and Simulated Organizations, 11:69-88.
- [7] Chan, K. and Sterling L. 2003. Specifying Roles within Agent-Oriented Software Engineering. Proceedings 10th Asia-Pacific Software Engineering Conference, pp. 390-395.
- [8] Chien, S., Sherwood, R., Rabideau, G., Castano, R., Davies, A., Burl, M., Knight, R., Stough, T., Roden, J., Zetocha, P., Wainwright, R., Klupar, P., Van Gaasbeck,

- J. Cappelaere, P. and Oswald, D. 2002. The TechSat-21 Autonomous Space Science Agent. Proceedings 1st International Conference on Autonomous Agents, pp. 570-577.
- [9] Clark, P., Curtis, S. and Rilee, M. 2002. ANTS: Applying a New Paradigm to Lunar and Planetary Exploration. Proceedings Solar System Remote Sensing Symposium, Pittsburgh, PA, pp. 15-16.
- [10] Clark, P. E., Rilee, M. L., Curtis, S.A. and Cheung, C. 2004. In Situ Surveying of Saturn's Rings. Proceedings 35th Lunar and Planetary Science Conference, League City, Texas.
- [11] Clements, P. 2002. Being Proactive Pays Off. IEEE Software, 19(4):28, 30.
- [12] Clements, P. and Northrop, L. 2002. Software Product Lines. Boston: Addison-Wesley.
- [13] Coppit, D. and Sullivan, K.J. 2003. Sound Methods and Effective Tools for Engineering Modeling and Analysis. Proceedings 25th International Conference on Software Engineering, Portland, OR, pp. 198-207.
- [14] Curtis, S., Truskowski, W., Rilee, M. and Clark, P. 2003. ANTS for the Human Exploration and Development of Space. Proceedings IEEE Aerospace Conference, Big Sky, MT.
- [15] Curtis, S., Rilee, M., Clark, P. and Marr, G. 2003. Use of Swarm Intelligence in Spacecraft Constellations for the Resource Exploration of the Asteroid Belt. Proceedings 3rd International Workshop on Satellite Constellations and Formation Flying, Pisa, Italy.
- [16] Das, S., Krikorian, R. and Truskowski, W. 1999. Distributed Planning and Scheduling for Enhancing Spacecraft Autonomy. Proceedings 3rd Conference on Autonomous Agents, Seattle, WA, pp. 422-423.

- [17] Dehlinger, J. and Lutz, R. R. 2004. Software Fault Tree Analysis for Product Lines. Proceedings 8th IEEE International Symposium on High Assurance Systems Engineering, Tampa, FL, pp. 12-21.
- [18] Dehlinger, J. and Lutz, R. R. 2005. Applying Product-Line Fault Tree Analysis to Build Safer Multi-Agent Systems. Fast Abstract, 16th IEEE International Symposium on Software Reliability Engineering, Chicago, IL.
- [19] Dehlinger, J. and Lutz, R. R. 2005. A Product-Line Approach to Safe Reuse in Multi-Agent Systems. Proceedings 4th International Workshop on Software Engineering Large-Scale Multi-Agent Systems, St. Louis, MO, pp. 83-89.
- [20] Dehlinger, J., Feng, Q. and Hu, L. 2006. SSVChecker: Unifying Static Security Vulnerability Detection Tools in an Eclipse Plug-In. Proceedings 2006 OOPSLA Workshop Eclipse Technology Exchange Workshop at OOPSLA 2006, Portland, OR, pp. 30-34.
- [21] Dehlinger, J. and Lutz, R. R. 2006. A Product-Line Approach to Promote Asset Reuse in Multi-Agent Systems. Software Engineering for Multi-Agent Systems IV, Lecture Notes in Computer Science 3914, pp. 161-178.
- [22] Dehlinger, J. and Lutz, R. R. 2006. Bi-Directional Safety Analysis for Product-Line, Multi-Agent Systems. ACM SIGBED Review: Special Issues on Workshop Innovative Techniques for Certification of Embedded Systems, 3(4).
- [23] Dehlinger, J., Humphrey, M., Padmanabahn, P. and Lutz, R. R. 2007. Decimal and PLFaultCAT: From Product-Line Requirements to Product-Line Member Software Fault Trees. Proceedings 29th International Conference on Software Engineering, Minneapolis, MN, pp. 49-50.
- [24] Dehlinger, J. and Lutz, R. R. 2006. PLFaultCAT: A Product-Line Software Fault Tree Analysis Tool. Automated Software Engineering Journal, 13(1):169-193.

- [25] DeLoach, S. A. 2004. The MaSE Methodology. Methodologies and Software Engineering for Agent Systems-The Agent-Oriented Software Engineering Handbook Series: Multiagent Systems, Artificial Societies, and Simulated Organizations, 11:107-125.
- [26] Despotou, G. and Kelly, T. 2004. Extending the Safety Case Concept to Address Dependability. Proceedings 22nd International System Safety Conference, pp. 645-654.
- [27] Doerr, B. and Sharp, D. 2000. Freeing Product Line Architectures from Execution Dependencies. P. Donohoe ed., Proceedings Software Product-Line Engineering Conference, Kluwer Academic Publishers.
- [28] Doerr, J. 2002. Requirements Engineering for Product Lines: Guidelines for Inspecting Domain Model Relationships. Diploma Thesis, University of Kaiserslautern.
- [29] Douglass, B. P. 1999. Doing Hard Time: Developing Real-Time Systems with UML Objects, Frameworks and Patterns. Boston: Addison-Wesley.
- [30] Dugan, J. B. 2000. Galileo: A Tool for Dynamic Fault Tree Analysis. Proceedings 11th International Conference Computer Performance Evaluation: Modelling Technique and Tools, Schaumburg, IL, pp. 328-331.
- [31] European Cooperation for Space Standardization (ECSS). 2002. Space Engineering – Software, ECSS-E-40B (draft 1). http://esamultimedia.esa.int/docs/industry/SME/2003/software_engineering/materials/ecss-e-40b_draft1.pdf (Accessed June 2007).
- [32] Feng, Q. and Lutz, R. R. 2005. Bi-Directional Safety Analysis of Product Lines. Journal of Systems and Software, 78(2):111-127.
- [33] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

- [34] Giorgini, P., Kolp, M., Mylopoulos, J. and Pistore, M. 2004. The Tropos Methodology. *Methodologies and Software Engineering for Agent Systems-The Agent-Oriented Software Engineering Handbook Series: Multiagent Systems, Artificial Societies, and Simulated Organizations*, 11:89-106.
- [35] Girardi, R. 2002. Reuse in Agent-based Application Development. *Proceedings 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, Orlando, FL.
- [36] Gomaa, H. 2005. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architecture*. Addison-Wesley.
- [37] Hansen, K.M., Ravn, A.P. and Stavridou, V. 1998. From Safety Analysis to Software Requirements. *IEEE Transactions on Software Engineering*, 24(7):573-584.
- [38] Hara, H., Fujita, S. and Sugawara, K. 2000. Reusable Software Components Based on an Agent Model. *Proceedings Workshop on Parallel and Distributed Systems*, Iwate, Japan, pp. 447-452.
- [39] Heie, A. 2002. Global Software Product Lines and Infinite Diversity. http://www.sei.cmu.edu/SPLC2/keynote_slides/keynote_1.htm (Accessed May 2007).
- [40] Hollow, P., McDermid, J. and Nicholson, M. 2000. Approaches to Certification of Reconfigurable IMA Systems. *Proceedings 5th International Symposium of the International Council on Systems Engineering*.
- [41] Jennings, N. and Wooldridge, M. 2000. On Agent-Oriented Software Engineering. *Artificial Intelligence*, 117(2):277-296.
- [42] Juan, T. and Sterling, L. 2003. ROADMAP: Extending the Gaia Methodology for Complex Open Systems. *Proceedings 1st International Joint Conference Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, pp. 3-10.

- [43] Kang, K.C., Kim, S., Lee, J. and Lee, K. 1999. Feature-Oriented Engineering of PBX Software for Adaptability and Reusability. *Software Practice and Experience*, 29(10):167-177.
- [44] Leveson, N.G. 1995. *Safeware: System Safety and Computers*. Boston: Addison-Wesley.
- [45] Liu, J. 2007. Safety-Related Feature Interaction in Safety-Critical Product Lines. *Proceedings 29th International Conference Software Engineering Companion*, Minneapolis, MN, pp. 85-86.
- [46] Liu, J., Dehlinger, J. and Lutz, R. 2007. Safety Analysis of Software Product Lines Using State-Based Modeling. *Journal of Systems and Software*, to appear.
- [47] Liu, J., Dehlinger, J. and Lutz, R. 2005. Safety Analysis of Software Product Lines Using State-Based Modeling. *Proceedings 16th IEEE International Symposium on Software Reliability Engineering*, Chicago, IL, pp. 21-30.
- [48] Liu, J., Dehlinger, J., Sun, H. and Lutz, R. R. 2007. State-Based Modeling to Support the Evolution and Maintenance of Safety-Critical Software Product Lines. *Proceedings 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Tucson, AZ, pp. 596-608.
- [49] Lu, D. and Lutz, R.R. 2002. Fault Contribution Trees for Product Families. *Proceedings 13th International Symposium on Software Reliability Engineering*, Annapolis, MD, pp. 231-242.
- [50] Lutz, R. R. 2000. Extending the Product Family Approach to Support Safe Reuse. *Journal of Systems and Software*, 53(3):207-217.
- [51] Lutz, R. R. 2000. Software Engineering for Safety: A Roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, ACM Press.

- [52] Lutz, R. R. and Gannod, G. 2003. Analysis of a Software Product Line Architecture: An Experience Report. *The Journal of Systems and Software*. 66(3):253-267.
- [53] Lutz, R. R., Helmer, G. G., Moseman, M. M., Statezni, D. E. and Tockey, S. R. 1998. Safety Analysis of Requirements for a Product Family. Proceedings 3rd International Conference on Requirements Engineering, Colorado Springs, CO, pp. 24-31.
- [54] Lutz, R. R. and Woodhouse, R. M. 1999. Bi-Directional Analysis for Certification of Safety Critical Software. Proceedings 1st International Software Assurance Certification Conference.
- [55] Lutz, R. R. and Woodhouse, R.M. 1997. Requirements Analysis Using Forward and Backward Search. *Annals of Software Engineering*, 3:459-474.
- [56] NASA Strategic Roadmaps. NASA – APIO: Strategic Roadmaps - Overview, http://www.nasa.gov/about/strategic_roadmaps.html (Accessed May 2007).
- [57] Northrop, L. A Framework for Product Line Practice. Software Engineering Institute, <http://www.sei.cmu.edu/productlines/framework.html> (Accessed May 2007).
- [58] Padmanabhan, P. and Lutz, R.R. 2002. DECIMAL: A Requirements Engineering Tool for Product Families. Proceedings 2002 International Symposium Software Reliability Engineering for Product Lines, Essen, Germany, pp. 45-50.
- [59] Padmanabhan, P. and Lutz, R. R. 2005. Tool-Supported Verification of Product Line Requirements. *Automated Software Engineering Journal*, 12(4):447-465.
- [60] Pai, G. J. and Dugan, J. B. 2002. Automatic Synthesis of Dynamic Fault Trees from UML System Models. Proceedings 13th International Symposium on Software Reliability Engineering, Annapolis, MD, pp. 243-254.

- [61] Parnas, D. L. 1976. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, 2(1):193-213.
- [62] Pena, J., Dehlinger, J., Ruiz-Cortes, A., Hinchey, M. and Lutz, R. R. In press. Current Research in Multiagent System Product Lines (MAS-PL). *Agent-Oriented Software Engineering*.
- [63] Peña, J., Hinchey, M. and Cortés, A. 2006. Multi-Agent System Product Lines: Challenges and Benefits. *Communications of the ACM*, 49(12):82-84.
- [64] Peña, J., Hinchey, M., Cortés, A. and Trinidad, P. 2006. Building the Core Architecture of a NASA Multiagent System Product Line. *Proceedings 7th International ACM Workshop on Agent Oriented Software Engineering*, Hakodate, Japan, pp. 13-24.
- [65] Peña, J., Hinchey, M., Resinas, M., Sterritt, R. and Rash, J. 2006. Managing the Evolution of an Enterprise Architecture Using a MAS-Product-Line Approach. *Proceedings International Workshop on System/Software Architectures*, Las Vegas, NV, pp. 995-1001.
- [66] Peña, J., Hinchey, M. and Sterritt, R. 2006. Towards Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems Using an AOSE Methodology. *Proceedings 3rd IEEE International Workshop on Engineering of Autonomic and Autonomous Systems*, Columbia, MD, pp. 37-46.
- [67] Pohl, K., Bockle, G. and van der Linden, F. 2005. *Software Product-Line Engineering*. Berlin: Springer-Verlag.
- [68] Product Line Hall of Fame. Software Engineering Institute.
http://www.sei.cmu.edu/productlines/plp_hof.html (Accessed May 2007).
- [69] Radio Technical Commission for Aeronautics, RTCA/DO-178B: *Software Considerations in Airborne Systems and Equipment Certification*, 1992.

- [70] Rilee, M. and Stufflebeam, R. 2005. ANTS: Autonomous Nanotechnological Swarm. http://www.mind.ilstu.edu/curriculum/ants_nasa/ants_pam.php (Accessed May 2007).
- [71] Rouff, C., Hinchey, M., Rash, J., Truszkowski, W. and Sterritt, R. 2005. Towards Autonomic Management of NASA Missions. Proceedings 11th International Conference on Parallel and Distributed Systems, Fukuoka, Japan, pp. 473-477.
- [72] SAE, Aerospace Recommended Practice: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, ARP4761, 1996.
- [73] Schetter, T., Campbell, M. and Surka, D. 2000. Multiple Agent-Based Autonomy for Satellite Constellations. Proceedings 2nd International Symposium on Agent Systems and Applications, Zurich, Switzerland, pp. 147-180
- [74] Schmid, K. and Verlage, M. 2002. The Economic Impact of Product Line Adoption and Evolution. IEEE Software, 19(4):50-57.
- [75] Software Product Line Conferences. <http://www.splc.net/> (Accessed May 2007).
- [76] Sommerville, I. 2004. Software Engineering. Boston: Pearson Addison-Wesley.
- [77] Sterritt, R., Rouff, C., Rash, J., Truszkowski, W. and Hinchey, M. 2005. Self*-Properties in NASA Mission. Proceedings 2005 International Conference Software Engineering Research and Practice, Las Vegas, NV, pp. 66-72.
- [78] Sullivan, K. J., Dugan, J. B. and Coppit, D. 1999. The Galileo Fault Tree Analysis Tool. Proceedings 29th Annual International Symposium on Fault-Tolerant Computing, Madison, WI, pp. 232-235.
- [79] Sutandiyono, W., Chhetri, M. B., Krishnaswamy, S. and Loke, S. W. 2004. Experiences with Software Engineering of Mobile Agent Applications. Proceedings 2004 Australian Software Engineering Conference, pp. 339-349.

- [80] Svanberg, M., Gorp, J. and Bosch, J. 2005. A Taxonomy of Variability Realization Techniques. *Software – Practice & Experience*, 35(8):705-754.
- [81] Terrestrial Planet Finder Project. Planet Quest: Missions – Terrestrial Planet Finder. http://planetquest.jpl.nasa.gov/TPF/tpf_index.cfm (Accessed May 2007).
- [82] Toft, P., Coleman, D. and Ohta, J. 2000. A Cooperative Model for Cross-Divisional Product Development for a Software Product Line. In P. Donohoe ed., *Proceedings Software Product-Line Conference*, Kluwer Academic Publishers.
- [83] Truskowski, W. F., Hinchey, M. G., Rash, J. L. and Rouff, C. A. 2006. Autonomous and Autonomic Systems: A Paradigm for Future Space Exploration Missions. *IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews*, 36(3):279-291.
- [84] Truskowski, W., Rash, J., Rouff, C. and Hinchey, M. 2004. Asteroid Exploration with Autonomic Systems. *Proceedings 11th IEEE International Conference and Workshop Engineering of Computer-Based Systems*, Brno, Czech Republic, pp. 484-489.
- [85] Tveit, A. 2001. A Survey of Agent-Oriented Software Engineering. NTNU Computer Science Graduate Student Conference.
- [86] United States Department of Defense, Draft DoD Software Technology Strategy, Office of the Director, Defense Research & Engineering, DRAFT: December 1991.
- [87] van Ommering, R. 2005. Software Reuse in Product Populations. *IEEE Transactions on Software Engineering*, 31(7):537-550.
- [88] Weiss, D.M. and Lai, C. T. R. 1999. *Software Product Line Engineering: A Family-Based Software Development Process*. Boston: Addison-Wesley.
- [89] Wijinstra, J. 2002, Critical Factors for a Successful Platform-Based Product Family Approach. G. Chastek ed., *Proceedings Software Product-Line Engineering Conference*, Springer LNCS 2379.

- [90] Wooldridge, M. 1997. Agent-Based Software Engineering. *IEEE Proceedings on Software Engineering*, 144(1):26-37.
- [91] Wooldridge, M. and Jennings, N. 1994. Agent Theories, Architectures and Languages: A Survey. *Proceedings ECAI Workshop on Agent Theories, Architectures*, pp. 1-32.
- [92] Wooldridge, M., Jennings, N. and Kinny, D. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3): 285-312.
- [93] Yu, E. 1995. Modeling Strategic Relationships for Process Engineering. Ph.D. Thesis, University of Toronto.
- [94] Zambonelli, F., Jennings, R. and Wooldridge, M. 2003. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3): 317-370.

APPENDIX A. COMMONALITY AND VARIABILITY ANALYSIS

This appendix provides the full Commonality and Variability Analysis (CVA) for the Prospecting Asteroid Mission (PAM) multi-agent system product line (MAS-PL) case study used throughout this dissertation. The CVA provides a dictionary of relevant domain terms followed by the PAM case study's product-line commonality requirements and variability requirements.

DICTIONARY OF TERMS

- Agent*** In the case of *PAM* spacecraft, an agent is at the spacecraft-level comprising of all the roles that the spacecraft must perform.
- Altimeter*** Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's shape, 3D model, topography and *geomorphology* [68].
- ANTS*** The **A**utonomous **N**ano-**T**echnology **S**warm is a NASA concept mission that entails a grouping of *agents* that work *cooperatively*, *autonomously* and is adaptable to achieve mission goals [14], [15], [71], [83].
- AU*** **A**stronomical **U**nit, the approximate distance from the Sun to the Earth.
- Autonomous*** Systems that operate on their own to the maximum extent possible and require little to no human intervention or guidance [83].
- Cooperation*** The ability for spacecraft to work together to achieve mission goals [1], [14], [15], [59].
- Environment*** The surrounding space and conditions as well as the other *PAM* spacecraft.
- Formation Flying*** The necessity to orbit an asteroid in specified relative positions (in relation to the asteroid as well as other spacecraft) to obtain ideal conditions to perform scientific, communication and decision-making activities [1], [14], [15], [83], [84].
- Gamma-Ray Spectrometer*** Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's heavy element makeup and *volatile characterization* [68].
- Geomorphology*** The study of the landforms present on an asteroid including the landforms possible origin and evolution.

Lagrange Point Special regions of space where the gravity of the Moon, the Earth and the Sun balance such that a spacecraft can be parked there at the cost of using a relatively small amount of fuel [68].

Near-Infrared Spectrometer Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's mineral abundance mapping [68].

Neutral Mass Spectrometer Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's *volatile characterization* [68].

Neutron Spectrometer Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's heavy element makeup and *volatile characterization* [68].

PAM The **Prospecting Asteroid Mission**. A 2020-2025 proposed NASA submission lasting 5-10 years based on the *ANTS* technology with a goal of exploring the asteroid belt between Mars and Jupiter.

Photogeology The geologic interpretation of landforms on an asteroid via imaging.

Radio Science/Magnetometer Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's gravity and magnetic fields, interior makeup and 3D model [68].

Radio Sounder/Infrared Radiometer Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's *Regolith characterization* [68].

Regolith Characterization The characterization of the heterogeneous material covering the solid body of an asteroid.

Self-Coordination The ability of a *PAM* spacecraft, at both the system and individual level, to *automously* decide upon, assign and pursue scientific goals [83].

Self-Healing The ability of a *PAM* spacecraft, at both the system and individual level, to *autonomously* recover from damage due either to solar storms or collisions with an asteroid or other spacecraft [77].

Self-Optimization The ability of a *PAM* spacecraft, at both the system and individual level, to *autonomously* improve its ability to identify and explore asteroids through learning and experience. At the system level, optimization propagates upwards from the *self-optimization* of individuals [77].

Self-Protection The ability of a *PAM* spacecraft, at both the system and individual level, to *autonomously* protect itself from solar storms or collisions with an asteroid or other spacecraft [77].

Solar Storm Solar events that cause a large amount of solar radiation to be expelled from the Sun into space.

Subswarm A subset of *PAM* spacecraft.

Swarm The collection of all *PAM* spacecraft.

Visible Imager Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain a target asteroids detection, 3D model and photogeology [68].

Volatile Characterization The characterization of the volatile elements, those elements that vaporize at a relatively low temperature, present on an asteroid.

X-ray Spectrometer Scientific instrument onboard some *PAM* spacecraft with a primary task to obtain an asteroid's major element abundance mapping [68].

COMMONALITIES

General Commonality Requirements

- C_G1. The PAM swarm shall have no single point of failure [15].
- C_G2. The PAM swarm shall be robust to minor faults and catastrophic failures [14].

Self-Coordination Commonality Requirements

- C_SC1. Every spacecraft shall work cooperatively (in a hierarchical, social manner) with other spacecraft to achieve mission goals [64], [65], [66], [77], [83], [84].
- C_SC2. Every spacecraft shall be able to coordinate its own science operations while simultaneously maximizing the resource utilization [77].
- C_SC3. Every spacecraft shall have the ability to coordinate its own orbits and trajectories with others to avoid collisions [15], [71], [84].
- C_SC4. Every spacecraft shall have the capability of performing various kinds of formation flying [15].
- C_SC5. Every spacecraft shall be able to form subswarms under the control of a leader spacecraft [77], [83], [84].

Self-Healing Commonality Requirements

- C_SH1. Every spacecraft shall be able to recognize that its memory is corrupted/damaged (i.e., as a result from exposure to solar radiation) [64], [65], [66], [84].
- C_SH2. Every spacecraft shall be able to request an uncorrupted memory from another spacecraft in the event that it recognizes that its memory is corrupted [71], [84].

- C_SH3.** Every spacecraft shall be able to send its uncorrupted memory to another spacecraft upon request [71], [84].
- C_SH4.** Every spacecraft shall be able to request to be replaced by another spacecraft in the event that it recognizes that its memory has failed beyond repair [71], [84].
- C_SH5.** Every spacecraft shall be able to be killed off by a *leader* in the event of a loss of power [77], [84].

Self-Optimization Commonality Requirements

- C_SO1.** Every spacecraft shall be able to adjust to the surrounding environment (i.e., deteriorated/failing science instrumentation, when batteries/solar cells are deteriorating, etc.) [77], [84].
- C_SO2.** Every spacecraft shall be able to optimize itself through calibrating its instruments [77], [83], [84].
- C_SO3.** Every spacecraft shall be able to optimize its power consumption [15], [65], [66], [84].
- C_SO4.** Every spacecraft shall be able to monitor and adjust its relative positions to optimize its scientific exploration [77], [84].

Self-Protection Commonality Requirements

- C_SP1.** Every spacecraft shall be responsible for preventing collisions with other spacecraft [64], [65], [66], [71], [77], [84].
- C_SP2.** Every spacecraft shall be able to communicate with nearby spacecraft in order to prevent collisions [64], [65], [66], [71], [77], [84].
- C_SP3.** Every spacecraft shall be responsible for preventing collisions with asteroids [64], [65], [66], [71], [77], [84].

- C_SP4.** Every spacecraft shall be able to store a 3D map of nearby asteroids in order to prevent collisions [71], [77], [84].
- C_SP5.** Every spacecraft shall be able to take acceptable risks (i.e., collision with asteroids or other spacecraft) while attempting to satisfy its scientific goals [71], [77], [84].
- C_SP6.** Every spacecraft shall be able to deploy its solar sail to use as a shield for protection against solar storms [65], [66], [77], [83], [84].
- C_SP7.** Every spacecraft shall be able to switch off its subsystems when needed to protect against solar radiation [65], [66], [77], [83], [84].
- C_SP8.** Every spacecraft shall be able to receive messages from other spacecraft giving advanced warning of an impending solar storm [65], [66], [77], [84].

Miscellaneous Commonality Requirements

- C_M1.** Every spacecraft shall have the ability to control its own guidance navigation and control functions [14], [15], [83].
- C_M2.** Every spacecraft shall have the ability to control its own attitude [14], [15].
- C_M3.** Every spacecraft shall be able to use their solar shields as its means of flight [14], [15], [65], [66].
- C_M4.** Every spacecraft shall be able to know its current position [15], [65], [66].
- C_M5.** Every spacecraft shall be able to know its current velocity increment [15], [66].
- C_M6.** Every spacecraft shall be able to adjust its position/orbit [15], [65], [66].
- C_M7.** Every spacecraft shall be able to change its velocity increment [15], [65], [66].

- C_M8.** Every spacecraft shall be able to calculate the thrust needed to power its solar sails needed to maneuver [15], [65], [66].
- C_M9.** Every spacecraft shall be able to verify/check each other's results via a voting process (e.g., Byzantine voting schemes such as a 4-way or more may be needed) [15].

VARIABILITIES

General Variability Requirements

- V_G1.** Every spacecraft shall be initially defined by one of the roles it's to assume in the PAM swarm [14], [65], [66], [77], [83], [84].

Self-Coordination Variability Requirements

- * Self-coordination variability requirements are listed under the Leader, Messenger and/or Worker Variability Requirements, respectively.

Self-Healing Variability Requirements

- V_SH1.** A *messenger* spacecraft's ability to be upgraded to that of a *leader*'s role may vary [77].
- V_SH2.** A *leader* spacecraft's ability to be upgraded to that of a *messenger*'s role if a *messenger* is destroyed may vary [77].
- V_SH3.** A *worker* spacecraft's ability to be upgraded to that of a *messenger*'s role if a *messenger* is destroyed may vary [77].

Self-Optimization Variability Requirements

- V_SO1.** A spacecraft's ability to optimize itself via improving their ability to identify asteroids of interest may vary [15], [71], [77], [83] [84].

- V_SO2.** A spacecraft's ability to share its optimization information regarding the identification of asteroids of interest with *leader* spacecraft may vary [77], [84].
- V_SO3.** A spacecraft's ability to optimize itself through positioning itself appropriately to best facilitate communications with *messenger* spacecraft may vary [15], [77], [84].
- V_SO4.** A spacecraft's ability to share its optimization information regarding positioning itself appropriately to best facilitate communications with *messenger* spacecraft may vary [15], [77].
- V_SO5.** A spacecraft's ability to optimize itself via learning through their past experiences to better investigate an asteroid may vary [15], [77], [84].
- V_SO6.** A spacecraft's ability to share its optimization information regarding how to better investigate an asteroid with *worker* spacecraft may vary [15], [77], [84].

Self-Protection Variability Requirements

- V_SP1.** A spacecraft's ability to be tasked with constantly observing the solar disc to detect signs of an impending solar storm may vary [65], [66], [77], [84].
- V_SP2.** A spacecraft's ability to receive warnings from mission control of an impending solar storm may vary [65], [66], [77], [84].

Leader Spacecraft Variability Requirements

- V_L1.** A spacecraft's ability to be in charge of performing subswarm allocation and planning may vary [15], [71], [83], [84].
- V_L2.** A spacecraft performing subswarm allocation and planning may vary in its role in allocation and planning activities [15].

- V_L3.** A spacecraft's ability to be able to assign teams of *worker* and *messenger* spacecraft may vary [83].
- V_L4.** A spacecraft's ability to direct/coordinate *worker* spacecraft to investigate a specific asteroid may vary [77], [83], [84].
- V_L5.** A spacecraft's ability to redistribute/realign duties to worker spacecraft to ensure sufficient coverage of instrument roles may vary [83].
- V_L6.** A spacecraft's ability to be responsible for determining the types of asteroids to investigate may vary [15], [71], [77], [83], [84].
- V_L7.** A spacecraft's ability to contain the rules that decide the types of asteroids to investigate may vary [77], [83].
- V_L8.** A spacecraft's ability to be responsible for determining the types of data to gather from an asteroid may vary [77], [83].
- V_L9.** A spacecraft's ability to be able to decide amongst other leaders present in a subswarm which shall take the lead and control the subswarm may vary [15], [83].
- V_L10.** A spacecraft's ability to oversee the data flow from *worker* spacecraft to *messenger* spacecraft may vary [15].
- V_L11.** A spacecraft's ability to contain models of the types of science they want to have performed on a targeted asteroid may vary [15], [83].
- V_L12.** A spacecraft's ability to communicate to *messenger* spacecraft parts of the model of the science to be performed on a targeted asteroid may vary [15].
- V_L13.** A spacecraft's ability to form the communications layer to maintain the position, trajectory and orbital insertion data of every spacecraft in the swarm may vary [15], [83].
- V_L14.** A spacecraft's knowledge of the swarm may vary [15].

- V_L15.** A spacecraft's ability to receive and accept change in velocity bids from other members during subswarm reconfiguration may vary [15].
- V_L16.** A spacecraft's ability to issue a request to members of the subswarm for change in velocity bids may vary [15].
- V_L17.** A spacecraft's ability to issue a move to new position message to spacecraft of the subswarm during subswarm reconfiguration may vary [15].

Messenger Spacecraft Variability Requirements

- V_M1.** A spacecraft's ability to relay/coordinate messages between *worker* spacecraft and *leader* spacecraft may vary [15], [71], [77], [83], [84].
- V_M2.** A spacecraft's ability to relay/coordinate messages between *leader* spacecraft and mission control may vary [15] [71], [77].
- V_M3.** A spacecraft's ability to provide up to ~0.1 AU communication across the swarm may vary [15].
- V_M4.** A spacecraft's ability to receive asteroid data from *worker* spacecraft may vary [15], [71].
- V_M5.** A spacecraft's ability to archive data received from *worker* spacecraft regarding the discovered information of a targeted asteroid may vary [15].
- V_M6.** A spacecraft's ability to travel to a terrestrial Lagrange point (or other communication nodes) to communicate the discovered information may vary [14], [15], [83], [84].
- V_M7.** A spacecraft's ability to relay the parts of the model that a *leader* spacecraft wants *worker* spacecraft to carry out on a targeted asteroid to *worker* spacecraft may vary [15].

V_M8. A spacecraft's ability form the communications layer to maintain the position, trajectory and orbital insertion data of every spacecraft in the swarm may vary [14], [15], [83], [84].

Worker Spacecraft Variability Requirements

V_W1. A spacecraft's single onboard specialized scientific instrumentation may vary [1], [15], [65], [66], [68], [71], [77], [83], [84].

V_W2. A spacecraft's ability to communicate the data they have found regarding a targeted asteroid to the *messengers* may vary [77], [84].

V_W3. A spacecraft's ability to send asteroid data to a *messenger* spacecraft to be archived may vary [15].

V_W4. A spacecraft's ability to, when an opportunity presents itself, investigate a nearby asteroid to collect preliminary data so that it can be evaluated by a *leader* as to the level of interest the swarm should have for that particular asteroid may vary [1], [15], [68], [71], [83].

V_W5. A spacecraft's ability to work alone to evaluate potential asteroids to investigate may vary [15].

V_W6. A spacecraft equipped with visible imager instrumentation may vary in its field scope [15].

V_W7. A spacecraft equipped with visible imager instrumentation and containing the functionality to gather data related to asteroid target detection may vary [1], [15], [68], [71], [77], [83].

V_W8. A spacecraft equipped with visible imager instrumentation and containing the functionality to gather data in order to construction a 3D model of the target asteroid may vary [1], [15], [68], [71], [83].

- V_W9.** A spacecraft equipped with visible imager instrumentation and containing the functionality to gather data pertaining to the asteroid's photogeology may vary [1], [15], [68], [71], [83].
- V_W10.** A spacecraft equipped with visible imager instrumentation and containing the functionality to ascertain the exact location of a target asteroid may vary [1], [15], [68], [71], [83].
- V_W11.** A spacecraft equipped with visible imager instrumentation and containing the functionality to create a rough model of a target asteroid to be used by other worker spacecraft for maneuvering around the asteroid may vary [1], [15], [68], [71], [83].
- V_W12.** A spacecraft equipped with near-infrared spectrometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's mineral abundance mapping may vary [1], [15], [68], [71], [83].
- V_W13.** A spacecraft specialized with X-ray spectrometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's major element abundance mapping may vary [1], [15], [68], [71], [83].
- V_W14.** A spacecraft equipped with Gamma-ray instrumentation and containing the functionality to gather data pertaining to the target asteroid's heavy element abundance mapping may vary [1], [15], [68], [71], [83].
- V_W15.** A spacecraft specialized with Neutron spectrometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's volatile abundance mapping may vary [1], [15], [68], [71], [83].
- V_W16.** A spacecraft equipped with altimeter instrumentation and containing the functionality to gather data pertaining to the target asteroid's shape may vary [1], [15], [68], [71], [83].

- V_W17.** A spacecraft specialized with altimeter instrumentation and containing the functionality to gather data pertaining to the target asteroid's 3D model construction may vary [1], [15], [68], [71], [83].
- V_W18.** A spacecraft specialized with altimeter instrumentation and containing the functionality to gather data pertaining to the target asteroid's topography may vary [1], [15], [68], [71], [83].
- V_W19.** A spacecraft specialized with altimeter instrumentation and containing the functionality to gather data pertaining to the target asteroid's geomorphology may vary [1], [15], [68], [71], [83].
- V_W20.** A spacecraft specialized with radio science/magnetometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's gravity fields may vary [1], [15], [68], [71], [83].
- V_W21.** A spacecraft specialized with radio science/magnetometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's magnetic fields may vary [1], [15], [68], [71], [83].
- V_W22.** A spacecraft specialized with radio science/magnetometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's interior makeup may vary [1], [15], [68], [71], [83].
- V_W23.** A spacecraft specialized with radio science/magnetometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's 3D model construction may vary [1], [15], [68], [71], [83].
- V_W24.** A spacecraft specialized with radio sounder/infrared radiometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's Regolith characterization may vary [1], [15], [68], [71], [83].

V_W25. A spacecraft specialized with neutral mass spectrometer instrumentation and containing the functionality to gather data pertaining to the target asteroid's volatile characterization may vary [1], [15], [68], [71], [83].

APPENDIX B. PARAMETERS OF VARIATION

This appendix provides the Parameters of Variation tables for the Prospecting Asteroid Mission (PAM) multi-agent system product line (MAS-PL) case study used throughout this dissertation. The Parameters of Variation tables further define the product-line variability requirements detailed in the Commonality and Variability Analysis (CVA).

Parameter	Meaning	Domain	Binding Time	Default
GENERAL VARIABILITY REQUIREMENTS				
P1: vSpacecraftRole V_G1	The role that a spacecraft is to initially assume.	[Leader, Messenger, Worker]	Design	Worker
SELF-HEALING VARIABILITY REQUIREMENTS				
P2: vUpgradeToLeader V_SH1	The ability of a <i>messenger</i> spacecraft to be upgraded to assume the role of a <i>leader</i> .	[True, False]	Specification	False
P3: vUpgradeToMessenger V_SH2, V_SH3	The ability of a <i>leader</i> or a <i>worker</i> spacecraft to be upgraded to assume the role of a <i>messenger</i> .	[True, False]	Specification	False
SELF-OPTIMIZATION VARIABILITY REQUIREMENTS				
P4: vIdAsteroidsOptimization V_SO1, V_SO2	The ability of a <i>leader</i> spacecraft to optimize its ability to identify asteroids of interest and share this information with other <i>leader</i> spacecraft.	[True, False]	Specification	False
P5: vCommOptimization V_SO3, V_SO4	The ability of a spacecraft to optimize its positioning for communications and sharing this optimization with other spacecraft.	[True, False]	Specification	True
P6: vScienceOptimization V_SO5, V_SO6	The ability to optimize its scientific exploration of an asteroid and sharing this optimization with other spacecraft.	[True, False]	Specification	False
SELF-PROTECTION VARIABILITY REQUIREMENTS				
P7: vSolarDiscWatch V_SP1	The ability of a spacecraft to constantly watch the solar disc for the signs of an impending solar storm.	[Passive, Warm-Spare, Active]	Design	Passive
P8: vMissConStormWarn V_SP2	The ability of a spacecraft to receive messages from mission control warning of an impending solar storm.	[True, False]	Design	False

Parameter	Meaning	Domain	Binding Time	Default
LEADER SPACECRAFT VARIABILITY REQUIREMENTS				
P9: vAllocPlanAbility V_L1	The ability of a spacecraft to perform subswarm allocation and planning.	[True, False]	Specification	False
P10: vAllocPlanRole V_L2	The role of a spacecraft participating in subswarm allocation and planning.	[Passive, Active]	Runtime	Passive
P11: vAssignTeamsAbility V_L3	The ability to assign teams of <i>worker</i> and <i>messenger</i> spacecraft.	[True, False]	Specification	False
P12: vRedistribRolesAbility V_L4, V_L5	The ability to redistribute roles to <i>worker</i> spacecraft.	[True, False]	Specification	False
P13: vIdAsteroidAbility V_L6, V_L7, V_L8	The ability to be responsible for identifying which asteroids should be investigated by a subswarm.	[True, False]	Specification	False
P14: vDecideLeaderAbility V_L9	The ability to decide which leader should take lead control of a subswarm.	[True, False]	Specification	False
P15: vOverseeDataFlow V_L10	The ability to oversee the data flow from <i>worker</i> spacecraft to <i>messenger</i> spacecraft.	[True, False]	Specification	False
P16: vTargetAsteroidModel V_L11, V_L12	The ability to contain a model of the profile of the types of asteroids that should be explored and the ability to communicate this model with other spacecraft.	[True, False]	Specification	False
P17: vPosTrajOrbitDataHolder V_L13, V_M8	The ability to maintain the position, trajectory and orbital insertion data of every spacecraft in the subswarm.	[True, False]	Specification	False
P18: vLeaderSwarmKnow V_L14	The amount of knowledge that a spacecraft has about the entire swarm.	[Subswarm knowledge, Partial-swarm knowledge, Full-swarm knowledge]	Runtime	Subswarm knowledge
P19: vIdAsteroidsOptimization V_L15, V_L16, V_L17	The ability to facilitate and coordinate spacecraft during subswarm reconfiguration.	[True, False]	Specification	False

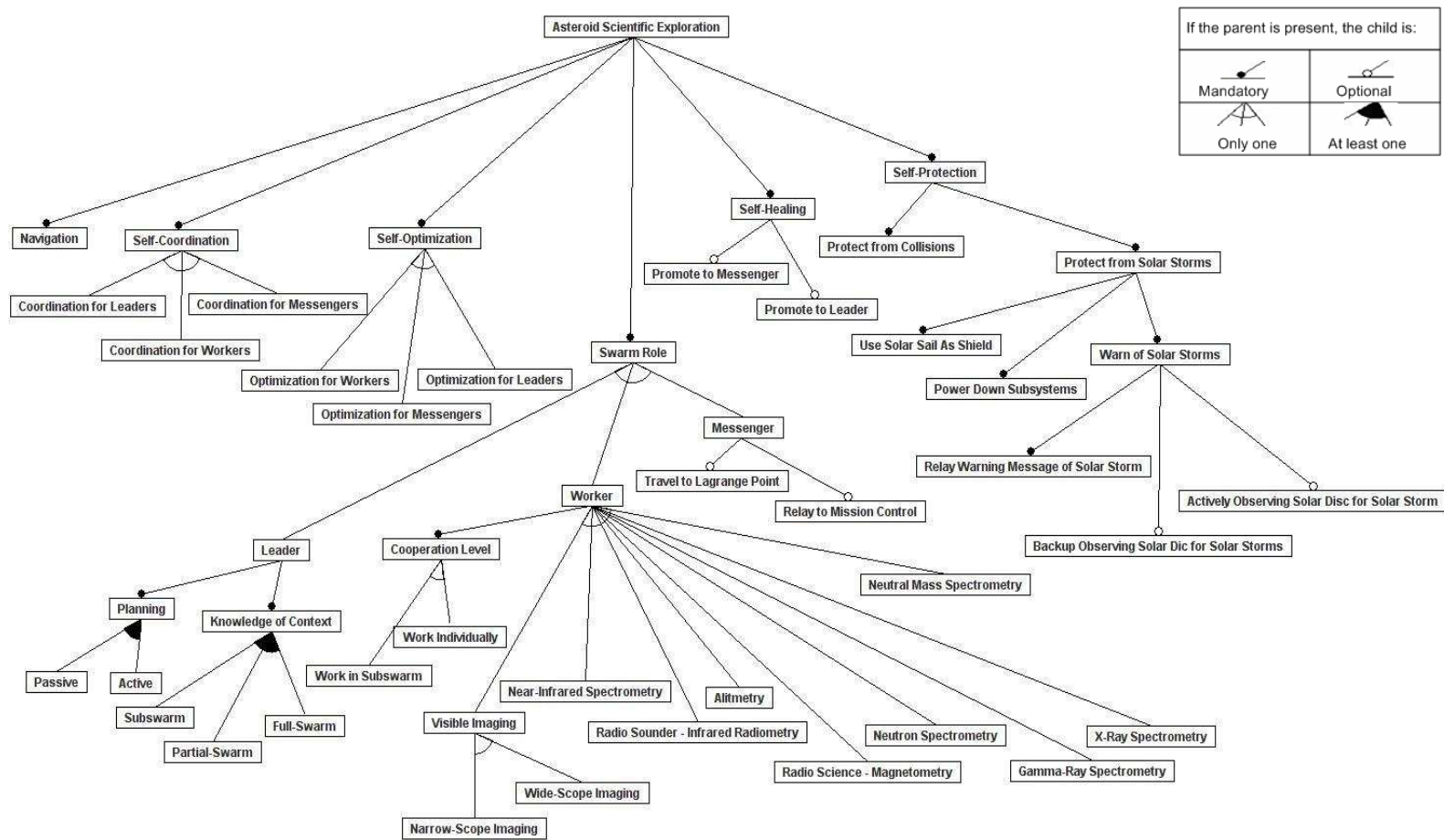
Parameter	Meaning	Domain	Binding Time	Default
MESSENGER SPACECRAFT VARIABILITY REQUIREMENTS				
P20: vRelayMessagesSwarm V_M1, V_M4	The ability to relay and coordinate messages between spacecraft.	[True, False]	Specification	False
P21: vRelayMessagesMisCon V_M2	The ability to relay and coordinate messages to mission control.	[True, False]	Specification	False
P22: vCommunicationRange V_M3	The range that a spacecraft can reliably communicate (in AU).	[0...0.1 AU]	Design	0.05 AU
P23: vArchiveAsteroidInfo V_M5	The ability to archive received data regarding the discovered information of a targeted asteroid.	[True, False]	Specification	False
P24: vTravelToLagrangePnt V_M6	The ability to travel to a Lagrange point to communicate with mission control.	[True, False]	Specification	False
P25: vRelayAsteroidModel V_M7	The ability to relay parts of the science model to carry out on a targeted asteroid to a <i>worker</i> spacecraft.	[True, False]	Specification	False
WORKER SPACECRAFT VARIABILITY REQUIREMENTS				
P26: vWorkerInstrument V_W1	The specialized scientific instrumentation that a spacecraft has onboard.	[Visible Imager, Near-Infrared Spectrometer, X-Ray Spectrometer, Gamma-Ray Spectrometer, Neutron Spectrometer, Altimeter, Radio Science/Magnetometer, Radio Sounder/Infrared Radiometer, Neutral Mass Spectrometer]	Design	Visible Imager
P27: vCommAsteroidData V_W2, V_W3	The ability to communicate data found regarding a targeted asteroid.	[True, False]	Specification	True
P28: vPreAsteroidInvestigate V_W4	The ability to preliminarily investigate a nearby asteroid for initial data when the opportunity presents itself.	[True, False]	Specification	True

Parameter	Meaning	Domain	Binding Time	Default
WORKER SPACECRAFT VARIABILITY REQUIREMENTS (continued)				
P29: vWorkAloneAbility V_W5	The ability for <i>worker</i> spacecraft to work alone rather than within a subswarm.	[True, False]	Specification	False
P30: vVisibleImagerScope V_W6	The field scope of a visible imager instrumentation.	[Narrow-Scope, Wide-Scope]	Design	Narrow-Scope
P31: vImagerGatherData V_W7	The ability of a spacecraft with a visible imager to gather data related to asteroid target detection.	[True, False]	Specification	False
P32: vImagerMake3DModel V_W8	The ability of a spacecraft with a visible imager to construct a 3D model of the target asteroid.	[True, False]	Specification	False
P33: vImagerPhotogeology V_W9	The ability of a spacecraft with a visible imager to gather data related to asteroid's photogeology.	[True, False]	Specification	False
P34: vImagerLocation V_W10	The ability of a spacecraft with a visible imager to determine location of an asteroid.	[True, False]	Specification	False
P35: vImagerManeuverModel V_W11	The ability of a spacecraft with a visible imager to create a model used for other spacecraft to maneuver around an asteroid.	[True, False]	Specification	False
P36: vNearInfSpecGatherData V_W12	The ability of a spacecraft with a near-infrared spectrometer to gather data pertaining to the target asteroid's mineral abundance mapping.	[True, False]	Specification	False
P37: vXRaySpecGatherData V_W13	The ability of a spacecraft with a X-ray spectrometer to gather data pertaining to the target asteroid's major element abundance mapping.	[True, False]	Specification	False
P38: vGammaRayGatherData V_W14	The ability of a spacecraft with a Gamma-ray instrument to gather data pertaining to the target asteroid's heavy element abundance mapping.	[True, False]	Specification	False

Parameter	Meaning	Domain	Binding Time	Default
WORKER SPACECRAFT VARIABILITY REQUIREMENTS (continued)				
P39: vNeutronSpecGatherData V_W15	The ability of a spacecraft with a Neutron spectrometer the target asteroid's volatile abundance mapping.	[True, False]	Specification	False
P40: vAltimeterGatherData V_W16	The ability of a spacecraft with an altimeter to gather data pertaining to the target asteroid's shape.	[True, False]	Specification	False
P41: vAltimeter3DModel V_W17	The ability of a spacecraft with an altimeter to construct a 3D model of the target asteroid.	[True, False]	Specification	False
P42: vAltimeterTopography V_W18	The ability of a spacecraft with an altimeter to gather data pertaining to the target asteroid's topography.	[True, False]	Specification	False
P43: vAltimeterGeomorphology V_W19	The ability of a spacecraft with an altimeter to gather data pertaining to the target asteroid's geomorphology.	[True, False]	Specification	False
P44: vRadScienceGatherData V_W20, V_W21	The ability of a spacecraft with a radio science/magnetometer to gather data pertaining to the target asteroid's gravity and magnetic fields.	[True, False]	Specification	False
P45: vRadScienceInterior V_W22	The ability of a spacecraft with a radio science/magnetometer to gather data regarding the asteroid's interior.	[True, False]	Specification	False
P46: vRadScience3DModel V_W23	The ability of a spacecraft with a radio science/magnetometer to gather data regarding the asteroid's 3D model.	[True, False]	Specification	False
P47: vRadSounderGatherData V_W24	The ability of a spacecraft with a radio sounder/infrared radiometer to gather data pertaining to the target asteroid's Regolith characterization.	[True, False]	Specification	False
P48: vNeutMassSpecGatherData V_W25	The ability of a spacecraft with a neutral mass spectrometer to gather data pertaining to the target asteroid's volatile characterization.	[True, False]	Specification	False

APPENDIX C. FEATURE MODEL

This appendix provides the full Feature Model for the Prospecting Asteroid Mission (PAM) multi-agent system product line (MAS-PL) case study used throughout this dissertation. The Feature Model illustrates the required and optional features of a PAM spacecraft based on the product-line commonality and variability requirements documented in the Commonality and Variability Analysis (CVA).



APPENDIX D. GAIA-PL ROLE SCHEMAS

This appendix provides the full set of Gaia-PL requirements specifications schemas for the Prospecting Asteroid Mission (PAM) multi-agent system product line (MAS-PL) case study used throughout this dissertation. The requirements specifications schemas further define the product-line commonality and variability requirements documented in the Commonality and Variability Analysis (CVA) and in the Parameters of Variation table.

Role Schema: Navigator	Schema ID: N
Variation Point: N/A	
Inherits: None	
Parameters of Variation: N/A	
Requirements: C_M1, C_M2, C_M3, C_M4, C_M5, C_M6, C_M7, C_M8	
Description: Provides the functionality to a spacecraft to maneuver itself using its solar sail.	
Activities and Protocols: AdjustSolarSail, CalculateThrust, CheckOrbit, CheckSolarSailStatus, CheckSystemStatus, ExtendSolarSail, MoveToPosition, RetractSolarSail	
Permissions:	
Reads -	
<i>currentAttitude</i>	// attitude of the spacecraft
<i>currentOrbit</i>	// current orbit of the spacecraft
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>systemStatus</i>	// status of the spacecraft
<i>solarSailStatus</i>	// status of the solar sail
Changes -	
<i>currentAttitude</i>	// attitude of the spacecraft
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
Generates -	
<i>systemStatus</i>	// status of the spacecraft
<i>solarSailStatus</i>	// status of the solar sail
<i>thrustNeeded</i>	// calculated thrust needed to move
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to maneuver the spacecraft to the desired location.	
Safety -	
None.	

Navigator Role Schema

Role Variation Points Schema: SelfCoordinator		Schemata ID: SC
Parameters of Variation: P9, P14, PP19, P20, P21		
Description:		
At the swarm-level, the collection of this role within all the spacecraft aid in autonomously coordinating the scientific pursuits of the spacecraft in the swarm. At the spacecraft-level, these roles aid in the spacecraft to individually decide the best way to achieve its given scientific goals and to communicate with nearby spacecraft to cooperate in achieving these goals.		
Variation Points:		
<u>Core:</u>	The core elements of a spacecraft to be able to autonomously coordinate itself and its surrounding spacecraft to decide upon, assign and pursue scientific goals. [SC-Core]	
Leader:	The elements needed in a <i>leader</i> spacecraft to be able to coordinate the subswarm spacecraft to pursue scientific goals. This includes coordinating with other <i>leader</i> spacecraft and coordinating all subswarm spacecraft during times of subswarm reconfiguration. [SC-Leader]	
Messenger:	The elements needed in a <i>messenger</i> spacecraft to be able to coordinate the communication network needed in a subswarm while pursuing scientific goals. [SC-Messenger]	
Worker:	The elements needed in a <i>worker</i> spacecraft to be able to coordinate the pursuit of science goals for a given asteroid. [SO-Worker]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time. However, a spacecraft that may switch its operating variation point (i.e., P2=True or P3=True) may have this variation point alter at runtime.		

Self-Coordinator Role Variation Points Schema

Role Schema: SelfCoordinator	Schema ID: SC-Core
Variation Point: Core	
Inherits: None	
Parameters of Variation: N/A	
Requirements: C_SC1, C_SC2, C_SC3, C_SC4, C_SC5, C_SP1, C_SP2, C_M1, C_M2, C_M4, C_M5, C_M6, C_M7, C_M8	
Description: Provides the spacecraft with the functionality to autonomously coordinate itself and its surrounding spacecraft to decide upon, assign and pursue scientific goals.	
Activities and Protocols: CalcOrbit, CalcPostion, CalcResourceUtil, CalcTrajectory, EvaluateCurrentGoal, JoinSubswarm, MoveNewPosition, PerformFormationFly, <u>AcceptFormFlyReq</u> , <u>AcceptSubswarmJoinReq</u> , <u>CoordinateOrbit</u> , <u>CoordinatePosition</u> , <u>CoordinateTrajectory</u> , <u>RejectFormFlyReq</u> , <u>RejectSubswarmJoinReq</u>	
Permissions:	
Reads -	
<i>spacecraftID</i>	// spacecraft ID to send to other spacecraft
<i>systemStatus</i>	// when requesting clean memory
<i>riskForSystemFactor</i>	// current status of the spacecraft
	// current risk to spacecraft to see if recent
	// solar storm
<i>systemGoal</i>	// current goal of the spacecraft
<i>currentAttitude</i>	// current attitude of the spacecraft
<i>currentGoal</i>	// current goal of the spacecraft
<i>currentPosition</i>	// current position of the spacecraft
<i>currentVelocityIncr</i>	// velocity increment of the spacecraft
<i>environmentStatus</i>	// current status of the detectable parts of
	// the surrounding environment
Changes -	
<i>currentAttitude</i>	// attitude of the spacecraft
<i>currentPosition</i>	// position of the spacecraft
<i>currentVelocityIncr</i>	// velocity increment of the spacecraft
<i>subswarmID</i>	// identification of newly joined subswarm
<i>subswarmSpacecraft</i>	// vector of other spacecraft in the newly
	// joined subswarm
Generates -	
<i>newSystemGoal</i>	// new goal of the spacecraft
<i>subswarmAcceptMsg</i>	// message to be sent accepting the
	// request to join a subswarm
<i>subswarmRejectMsg</i>	// message to be sent rejecting the
	// request to join a subswarm
<i>resourceUtilizationVal</i>	// calculated resource utilization level in
	// order to maximize science operations
	// and resource utilization
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will be able to coordinate its science operations and maximize its resource utilization.	
Safety -	
Avoiding collisions during formation flying via coordination.	

Core Variation Point Schema for the Self-Coordinator Role

Role Schema: SelfCoordinator	Schema ID: SC-Leader
Variation Point: Leader	
Inherits: SC-Core	
Parameters of Variation: P15=True; P16=True; P17=True; P18=Subswarm; P19=True	
Requirements: V_L10, V_L12, V_L13, V_L14, V_L15, V_L16, V_L17	
Description: Provides the core elements of a <i>leader</i> spacecraft to be able to facilitate the management and coordination of its subswarm.	
Activities and Protocols: CalculatePartModel, OverseeSubSwarmDataFlow, PerformSubswarmReconfig, AcceptSubswarmChangeVelocityBid, MoveNewPositionCom, ReqDataFlow, ReqSubswarmVelocityBids, SendModelPartMessenger, SendSubswarmVelocityBidConfirm	
Permissions: Reads - <i>subswarmSpacecraft</i> // vector of the spacecraft in the subswarm <i>subswarmSpacecraftPos</i> // vector of the all the spacecrafts current // positions in the subswarm <i>leaderSpacecraft</i> // vector of the <i>leader</i> spacecraft in the // subswarm supplied <i>velocityBidRec</i> // vector of received change of velocity bid Changes - <i>asteroidModel</i> // current model of an asteroid to send <i>subswarmSpacecraft</i> // vector of the spacecraft in the subswarm <i>subswarmSpacecraftPos</i> // vector of the all the spacecrafts positions // in the subswarm <i>leaderSpacecraft</i> // vector of the <i>leader</i> spacecraft in the // subswarm Generates - <i>partialAsteroidModel</i> // derived partial model to send to a // messenger so that spacecraft can avoid // collisions with asteroids	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the configuration and plans of the subswarm to achieve subswarm goals. Safety - Avoiding collisions by maintaining and coordinating spacecraft positions and movements.	

Leader Variation Point Schema for the Self-Coordinator Role

Role Schema: SelfCoordinator	Schema ID: SC-Messenger
Variation Point: Messenger	
Inherits: SC-Core	
Requirements: V_M1, V_M2, V_M4,	
Parameters of Variation: P20=True; P21=True	
Description: Provides the spacecraft with the elements needed in a <i>messenger</i> spacecraft to be able to coordinate the communication network needed in a subswarm while pursuing scientific goals.	
Activities and Protocols: CoordinateLeadToWorkMsg, CoordinateWorkToLeadMsg, <u>AcceptLeaderMsg</u> , <u>AcceptWorkerMsg</u> , <u>SendLeaderMsg</u> , <u>SendMsgMisCon</u> , <u>SendWorkerMsg</u>	
Permissions: Reads - <i>leaderSpacecraft</i> // vector of the <i>leader</i> spacecraft in the // subswarm <i>workerSpacecraft</i> // vector of the <i>worker</i> spacecraft in the // subswarm Changes - <i>currentGoal</i> // current goal of the spacecraft Generates - <i>missionControlMsg</i> // message to be sent to mission control // on behalf of a <i>leader</i> spacecraft	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensure the timely delivery of a message to be sent throughout the subswarm. Safety - None.	

Messenger Variation Point Schema for the Self-Coordinator Role

Role Schema: SelfCoordinator	Schema ID: SC-Worker
Variation Point: Worker	
Inherits: SC-Core	
Requirements: V_W2, V_W3	
Parameters of Variation: P27=True	
Description: The elements needed in a <i>worker</i> spacecraft to be able to coordinate the pursuit of science goals for a given asteroid.	
Activities and Protocols: CoordinateWorkerGoals, <u>AcceptAsteroidData</u> , <u>SendArchiveData</u> , <u>SendAsteroidData</u>	
Permissions:	
Reads -	
<i>asteroidID</i>	// identification of the current asteroid // under exploration
<i>asteriodData</i>	// current collected data of an asteroid
<i>currentGoal</i>	// current goal of the spacecraft
<i>workerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm
<i>likeWorkerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm with the same specialized // instrumentation
Changes -	
<i>asteriodData</i>	// collected data of an asteroid
Generates -	
<i>asteroidArchiveDataMsg</i>	// message to be sent containing the // data of an asteroid to be archived by a // <i>messenger</i> spacecraft
<i>asteriodDataMsg</i>	// message to be sent containing the // current collected data of an asteroid
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to report the data collected of a specific asteroid.	
Safety -	
None.	

Worker Variation Point Schema for the Self-Coordinator Role

Role Variation Points Schema: SelfOptimizer		Schemata ID: SO
Parameters of Variation: P4, P5, P6		
Description:		
At the swarm-level, the collection of these roles within all the spacecraft aid in autonomously and continuously improving the spacecraft's ability to identify, explore and communicate the information discovered while investigating asteroids. At the spacecraft-level, these roles aid in the spacecraft to continuously learn and improve its specialized abilities and communicate its findings with other similar spacecraft.		
Variation Points:		
<u>Core:</u>	The core elements of a spacecraft to be able to optimize itself in regards to general spacecraft functions so that it can continuously learn from the environment and perform better within the swarm. [SO-Core]	
Leader:	The elements needed in a <i>leader</i> spacecraft to be able to optimize itself in regards to its ability to best manage, oversee and direct the swarm to optimize the swarm's ability to achieve scientific goals. [SO-Leader]	
Messenger:	The elements needed in a <i>messenger</i> spacecraft to be able to optimize itself in regards to its ability to best perform the communication necessary within the swarm so that commands and information can best be transmitted. [SO-Messenger]	
Worker:	The elements needed in a <i>worker</i> spacecraft to be able to optimize itself in regards to its ability to best optimize its ability to achieve its own scientific goals. [SO-Worker]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time. However, a spacecraft that may switch is operating variation point (i.e., P2=True or P3=True) may have this variation point alter at runtime.		

Self-Optimizer Role Variation Points Schema

Role Schema: SelfOptimizer	Schema ID: SO-Core																																
Variation Point: Core																																	
Inherits: None																																	
Parameters of Variation: N/A																																	
Requirements: C_SO1, C_SO2, C_SO3, C_SO4, C_M1, C_M2, C_M4, C_M5																																	
Description: Provides the spacecraft with the functionality to optimize itself in regards to general spacecraft functions so that it can continuously learn from the environment and perform better within the swarm.																																	
Activities and Protocols: AdjustToEnviron, CalcNewPosition, CalibrateInstr, CheckSystemStatus, CheckEnvironStatus, CheckPowerConsump, CheckSolarCellStatus, EvaluatePositionForGoal, MoveNewPos																																	
Permissions: Reads - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>currentAttitude</i></td> <td style="padding-left: 20px;">// current attitude of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentGoal</i></td> <td style="padding-left: 20px;">// current goal of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentPosition</i></td> <td style="padding-left: 20px;">// current position of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentVelocityIncr</i></td> <td style="padding-left: 20px;">// current velocity increment of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>environmentStatus</i></td> <td style="padding-left: 20px;">// current status of the detectable parts of the surrounding environment</td> </tr> <tr> <td style="padding-left: 20px;"><i>powerConsumpLevel</i></td> <td style="padding-left: 20px;">// current level of the spacecraft's power consumption</td> </tr> <tr> <td style="padding-left: 20px;"><i>riskForSystemFactor</i></td> <td style="padding-left: 20px;">// current risk to spacecraft to see if recent solar storm</td> </tr> <tr> <td style="padding-left: 20px;"><i>solarCellLevel</i></td> <td style="padding-left: 20px;">// current status level of the spacecraft's solar cells</td> </tr> <tr> <td style="padding-left: 20px;"><i>systemStatus</i></td> <td style="padding-left: 20px;">// current status of the spacecraft</td> </tr> </table> Changes - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>environmentState</i></td> <td style="padding-left: 20px;">// current state that the spacecraft believes its surrounding environment is in</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentPosition</i></td> <td style="padding-left: 20px;">// current position of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>currentAttitude</i></td> <td style="padding-left: 20px;">// current attitude of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>instrCalibValue</i></td> <td style="padding-left: 20px;">// vector of the current calibration values for the onboard instruments</td> </tr> <tr> <td style="padding-left: 20px;"><i>instrVector</i></td> <td style="padding-left: 20px;">// vector of all the spacecraft's onboard instruments</td> </tr> </table> Generates - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>newEnvironStatus</i></td> <td style="padding-left: 20px;">// new status of the detectable parts of the surrounding environment</td> </tr> <tr> <td style="padding-left: 20px;"><i>newVelocityIncr</i></td> <td style="padding-left: 20px;">// calculated new velocity increment for the spacecraft</td> </tr> </table>		<i>currentAttitude</i>	// current attitude of the spacecraft	<i>currentGoal</i>	// current goal of the spacecraft	<i>currentPosition</i>	// current position of the spacecraft	<i>currentVelocityIncr</i>	// current velocity increment of the spacecraft	<i>environmentStatus</i>	// current status of the detectable parts of the surrounding environment	<i>powerConsumpLevel</i>	// current level of the spacecraft's power consumption	<i>riskForSystemFactor</i>	// current risk to spacecraft to see if recent solar storm	<i>solarCellLevel</i>	// current status level of the spacecraft's solar cells	<i>systemStatus</i>	// current status of the spacecraft	<i>environmentState</i>	// current state that the spacecraft believes its surrounding environment is in	<i>currentPosition</i>	// current position of the spacecraft	<i>currentAttitude</i>	// current attitude of the spacecraft	<i>instrCalibValue</i>	// vector of the current calibration values for the onboard instruments	<i>instrVector</i>	// vector of all the spacecraft's onboard instruments	<i>newEnvironStatus</i>	// new status of the detectable parts of the surrounding environment	<i>newVelocityIncr</i>	// calculated new velocity increment for the spacecraft
<i>currentAttitude</i>	// current attitude of the spacecraft																																
<i>currentGoal</i>	// current goal of the spacecraft																																
<i>currentPosition</i>	// current position of the spacecraft																																
<i>currentVelocityIncr</i>	// current velocity increment of the spacecraft																																
<i>environmentStatus</i>	// current status of the detectable parts of the surrounding environment																																
<i>powerConsumpLevel</i>	// current level of the spacecraft's power consumption																																
<i>riskForSystemFactor</i>	// current risk to spacecraft to see if recent solar storm																																
<i>solarCellLevel</i>	// current status level of the spacecraft's solar cells																																
<i>systemStatus</i>	// current status of the spacecraft																																
<i>environmentState</i>	// current state that the spacecraft believes its surrounding environment is in																																
<i>currentPosition</i>	// current position of the spacecraft																																
<i>currentAttitude</i>	// current attitude of the spacecraft																																
<i>instrCalibValue</i>	// vector of the current calibration values for the onboard instruments																																
<i>instrVector</i>	// vector of all the spacecraft's onboard instruments																																
<i>newEnvironStatus</i>	// new status of the detectable parts of the surrounding environment																																
<i>newVelocityIncr</i>	// calculated new velocity increment for the spacecraft																																
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the spacecraft's ability to perform its given tasks.																																	
Safety - None.																																	

Core Variation Point Schema for the Self-Optimizer Role

Role Schema: SelfOptimizer	Schema ID: SO-Leader
Variation Point: Leader	
Inherits: SO-Core	
Requirements: V_SO1, V_SO2, C_SC1, C_SC2, V_L6, V_L7, V_L8, V_L11	
Parameters of Variation: P4=True	
Description: Provides the spacecraft with the he elements needed in a <i>leader</i> spacecraft to be able to optimize itself in regards to its ability to best manage, oversee and direct the swarm to optimize the swarm's ability to achieve scientific goals. Specifically, the ability for a <i>leader</i> spacecraft to optimize its ability to identify asteroids of interest and share this information.	
Activities and Protocols: DeviseNewAsteroidIdRules, EvaluateCurrentAsteroidIdRules, ReviewAsteroidIdHis, <u>AcceptOptimizationInfo</u> , <u>AcceptOptimizationReq</u> , RequestOptimizationInfo, <u>ShareOptimizationInfo</u>	
Permissions: Reads - <ul style="list-style-type: none"> <i>asteroidIdRules</i> // current vector of rules that is used to identify asteroids of interest given preliminary data points on the asteroid <i>asteroidPrelimData</i> // preliminary data points of an asteroid <i>asteroidId</i> // identification number of an asteroid <i>asteroidIdHistory</i> // the history log kept of the spacecraft's identification of asteroids of interest <i>optimizationInfoRec</i> // message to received after requesting for another spacecraft's current optimization information <i>leaderVector</i> // vector of nearby <i>leader</i> spacecraft to aid in sharing optimization information Changes - <ul style="list-style-type: none"> <i>asteroidIdRules</i> // vector of rules that is used to identify asteroids of interest given preliminary data points on the asteroid Generates - <ul style="list-style-type: none"> <i>asteroidIdRulesValue</i> // evaluation value of the accuracy of the spacecraft's current ability to identify asteroids of interest <i>optimizationInfoMsg</i> // message to deliver upon receiving a request for spacecraft's current optimization information 	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to identify asteroids of interests to investigate for all <i>leader</i> spacecraft in the swarm. Safety - None.	

Leader Variation Point Schema for the Self-Optimizer Role

Role Schema: SelfOptimizer	Schema ID: SO-Messenger														
Variation Point: Messenger															
Inherits: SO-Core															
Requirements: V_SO3, V_SO4, C_SC1, C_SC2															
Parameters of Variation: P5=True															
<p>Description: Provides the spacecraft with the elements needed in a <i>messenger</i> spacecraft to be able to optimize itself in regards to its ability to best perform the communication necessary within the swarm so that commands and information can best be transmitted. Specifically, the ability of the spacecraft to optimize its positioning for communications and sharing this information with others.</p>															
<p>Activities and Protocols: DeviseNewCommStrategy, EvaluateCurrentCommStrategy, EvaluateCurPosition, ReviewCommHis, <u>AcceptOptimizationInfo</u>, <u>AcceptOptimizationReq</u>, <u>RequestOptimizationInfo</u>, <u>ShareOptimizationInfo</u></p>															
<p>Permissions:</p> <p>Reads -</p> <table> <tr> <td><i>communicationStrategy</i></td> <td>// current strategy for spacecraft's // communication</td> </tr> <tr> <td><i>communicationHist</i></td> <td>// current history log of the spacecraft's // past communication sessions</td> </tr> <tr> <td><i>optimizationInfoRec</i></td> <td>// message to received after requesting // for another spacecraft's current // optimization information</td> </tr> <tr> <td><i>messengerVector</i></td> <td>// vector of nearby <i>messenger</i> spacecraft // to aid in sharing optimization information</td> </tr> </table> <p>Changes -</p> <table> <tr> <td><i>communicationStrategy</i></td> <td>// current strategy for spacecraft's // communication</td> </tr> </table> <p>Generates -</p> <table> <tr> <td><i>optimizationInfoMsg</i></td> <td>// message to deliver upon receiving a // request for spacecraft's current // optimization information</td> </tr> <tr> <td><i>communicationStratVal</i></td> <td>// evaluation value of the accuracy of the // spacecraft's current ability to // communicate with the subswarm</td> </tr> </table>		<i>communicationStrategy</i>	// current strategy for spacecraft's // communication	<i>communicationHist</i>	// current history log of the spacecraft's // past communication sessions	<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information	<i>messengerVector</i>	// vector of nearby <i>messenger</i> spacecraft // to aid in sharing optimization information	<i>communicationStrategy</i>	// current strategy for spacecraft's // communication	<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information	<i>communicationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // communicate with the subswarm
<i>communicationStrategy</i>	// current strategy for spacecraft's // communication														
<i>communicationHist</i>	// current history log of the spacecraft's // past communication sessions														
<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information														
<i>messengerVector</i>	// vector of nearby <i>messenger</i> spacecraft // to aid in sharing optimization information														
<i>communicationStrategy</i>	// current strategy for spacecraft's // communication														
<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information														
<i>communicationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // communicate with the subswarm														
<p>Responsibilities:</p> <p>Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to communicate for all <i>messenger</i> spacecraft in the swarm.</p> <p>Safety - None.</p>															

Messenger Variation Point Schema for the Self-Optimizer Role

Role Schema: SelfOptimizer	Schema ID: SO-Worker																				
Variation Point: Worker																					
Inherits: SO-Core																					
Requirements: V_SO5, V_SO6, C_SC1, C_SC2																					
Parameters of Variation: P6=True																					
Description: The elements needed in a <i>worker</i> spacecraft to be able to optimize itself in regards to its ability to best optimize its ability to achieve its own scientific goals.																					
Activities and Protocols: DeviseNewSciExplorStrategy, EvaluateCurrentSciExplorStrategy, EvaluateCurPosition, ReviewSciExplorHis, <u>AcceptOptimizationInfo</u> , <u>AcceptOptimizationReq</u> , <u>RequestOptimizationInfo</u> , <u>ShareOptimizationInfo</u>																					
Permissions: Reads - <table border="0" style="width: 100%;"> <tr> <td style="padding-left: 20px;"><i>optimizationInfoRec</i></td> <td style="padding-left: 20px;">// message to received after requesting // for another spacecraft's current // optimization information</td> </tr> <tr> <td style="padding-left: 20px;"><i>sciExplorationStrategy</i></td> <td style="padding-left: 20px;">// current strategy for spacecraft's // science exploration using its specialized // onboard equipment</td> </tr> <tr> <td style="padding-left: 20px;"><i>sciExplorationRules</i></td> <td style="padding-left: 20px;">// current rules for the spacecraft to abide // by in its scientific exploration</td> </tr> <tr> <td style="padding-left: 20px;"><i>sciExplorationHist</i></td> <td style="padding-left: 20px;">// current history log of the spacecraft's // past science exploration of asteroids</td> </tr> <tr> <td style="padding-left: 20px;"><i>workerType</i></td> <td style="padding-left: 20px;">// the type of worker spacecraft (i.e., based // on its specialized onboard equipment</td> </tr> <tr> <td style="padding-left: 20px;"><i>workerVector</i></td> <td style="padding-left: 20px;">// vector of nearby <i>worker</i> spacecraft with // the same onboard equipment</td> </tr> <tr> <td style="padding-left: 20px;"><i>scienceGoal</i></td> <td style="padding-left: 20px;">// current scientific goal pursued by the // spacecraft</td> </tr> </table> Changes - <table border="0" style="width: 100%;"> <tr> <td style="padding-left: 20px;"><i>sciExplorationStrategy</i></td> <td style="padding-left: 20px;">// strategy for spacecraft's science // exploration using its specialized onboard // equipment</td> </tr> </table> Generates - <table border="0" style="width: 100%;"> <tr> <td style="padding-left: 20px;"><i>optimizationInfoMsg</i></td> <td style="padding-left: 20px;">// message to deliver upon receiving a // request for spacecraft's current // optimization information</td> </tr> <tr> <td style="padding-left: 20px;"><i>sciExplorationStratVal</i></td> <td style="padding-left: 20px;">// evaluation value of the accuracy of the // spacecraft's current ability to // achieve its scientific goals</td> </tr> </table>		<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information	<i>sciExplorationStrategy</i>	// current strategy for spacecraft's // science exploration using its specialized // onboard equipment	<i>sciExplorationRules</i>	// current rules for the spacecraft to abide // by in its scientific exploration	<i>sciExplorationHist</i>	// current history log of the spacecraft's // past science exploration of asteroids	<i>workerType</i>	// the type of worker spacecraft (i.e., based // on its specialized onboard equipment	<i>workerVector</i>	// vector of nearby <i>worker</i> spacecraft with // the same onboard equipment	<i>scienceGoal</i>	// current scientific goal pursued by the // spacecraft	<i>sciExplorationStrategy</i>	// strategy for spacecraft's science // exploration using its specialized onboard // equipment	<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information	<i>sciExplorationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // achieve its scientific goals
<i>optimizationInfoRec</i>	// message to received after requesting // for another spacecraft's current // optimization information																				
<i>sciExplorationStrategy</i>	// current strategy for spacecraft's // science exploration using its specialized // onboard equipment																				
<i>sciExplorationRules</i>	// current rules for the spacecraft to abide // by in its scientific exploration																				
<i>sciExplorationHist</i>	// current history log of the spacecraft's // past science exploration of asteroids																				
<i>workerType</i>	// the type of worker spacecraft (i.e., based // on its specialized onboard equipment																				
<i>workerVector</i>	// vector of nearby <i>worker</i> spacecraft with // the same onboard equipment																				
<i>scienceGoal</i>	// current scientific goal pursued by the // spacecraft																				
<i>sciExplorationStrategy</i>	// strategy for spacecraft's science // exploration using its specialized onboard // equipment																				
<i>optimizationInfoMsg</i>	// message to deliver upon receiving a // request for spacecraft's current // optimization information																				
<i>sciExplorationStratVal</i>	// evaluation value of the accuracy of the // spacecraft's current ability to // achieve its scientific goals																				
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to achieve scientific goals for all similar <i>worker</i> spacecraft in the swarm. Safety - None.																					

Worker Variation Point Schema for the Self-Optimizer Role

Role Variation Points Schema: LeaderPlanner		Schemata ID: LP
Parameters of Variation: P20, P21, P22, P23, P24, P25		
Description:		
Provides the <i>leader</i> spacecraft with the functionality to be able to manage, plan and coordinate the spacecraft of a subswarm to pursue and satisfy system-wide and individual scientific goals.		
Variation Points:		
<u>Passive:</u>	The elements of a passive <i>leader</i> spacecraft (i.e., a spacecraft acting as a backup to double-check all commands and calculations pertaining to the planning for the subswarm) to be able to manage, plan and coordinate the spacecraft of a subswarm. [LP-Passive]	
Active:	The elements of an active <i>leader</i> spacecraft (i.e., a spacecraft actively in charge) to be able to manage, plan and coordinate the spacecraft of a subswarm. [LP-Active]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time. All spacecraft shall have the Passive variation point as a commonality. Spacecraft with the Active variation point shall also include all functionality of Passive and may switch its variation point at runtime.		

Leader Planner Role Variation Points Schema

Role Schema: LeaderPlanner	Schema ID: LP- Passive																																																		
Variation Point: LeaderPlanner																																																			
Inherits: SC-Core																																																			
Parameters of Variation: P9=True; P10=Passive; P11=True; P12=True; P13=True; P14=True																																																			
Requirements: V_L1, V_L2, V_L3, V_L4, V_L5, V_L6, V_L7, V_L8, V_L9, V_L11																																																			
Description: Provides the spacecraft with the he elements needed in a <i>leader</i> spacecraft to be able to be able to passively (i.e., act as a backup) coordinate/plan the subswarm spacecraft to pursue scientific goals.																																																			
Activities and Protocols: CheckDecideDataToGather, CheckSubswarmAlloc, CheckSubswarmPlan, AcceptPlanToCheck, VoteLeaderElection																																																			
Permissions: Reads - <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>leaderSpacecraft</i></td> <td style="padding-left: 20px;">// vector of the <i>leader</i> spacecraft in the</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// subswarm</td> </tr> <tr> <td style="padding-left: 20px;">supplied <i>allocationStrategy</i></td> <td style="padding-left: 20px;">// supplied allocation strategy for the sub-</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// swarm to perform scientific exploration</td> </tr> <tr> <td style="padding-left: 20px;">supplied <i>scienceRules</i></td> <td style="padding-left: 20px;">// supplied rules used to investigate the</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// types of asteroids to explore</td> </tr> <tr> <td style="padding-left: 20px;">supplied <i>plan</i></td> <td style="padding-left: 20px;">// supplied plan for the subswarm to</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// achieve system-wide scientific goals</td> </tr> </table> Changes - <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>allocationStrategy</i></td> <td style="padding-left: 20px;">// allocation strategy for the subswarm to</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// perform scientific exploration</td> </tr> <tr> <td style="padding-left: 20px;"><i>plan</i></td> <td style="padding-left: 20px;">// plan for the subswarm to achieve</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// system-wide scientific goals</td> </tr> <tr> <td style="padding-left: 20px;"><i>scienceRules</i></td> <td style="padding-left: 20px;">// rules used to investigate the types of</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// asteroids to explore</td> </tr> <tr> <td style="padding-left: 20px;"><i>subswarmSpacecraft</i></td> <td style="padding-left: 20px;">// vector of the spacecraft in the subswarm</td> </tr> <tr> <td style="padding-left: 20px;"><i>leaderSpacecraft</i></td> <td style="padding-left: 20px;">// vector of the <i>leader</i> spacecraft in the</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// subswarm</td> </tr> </table> Generates - <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>allocationStratMsg</i></td> <td style="padding-left: 20px;">// newly devised subswarm allocation</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// strategy message to be sent out to</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// <i>leader</i> agreeing with or disagreeing with</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// the newly devised allocation strategy</td> </tr> <tr> <td style="padding-left: 20px;"><i>planMsg</i></td> <td style="padding-left: 20px;">// newly devised subswarm plan to achieve</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// system-wide scientific goals</td> </tr> <tr> <td style="padding-left: 20px;"><i>leaderElecVoteMsg</i></td> <td style="padding-left: 20px;">// message to be sent to vote for the ruler</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// of the <i>leader</i> spacecraft for a subswarm</td> </tr> </table>		<i>leaderSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the		// subswarm	supplied <i>allocationStrategy</i>	// supplied allocation strategy for the sub-		// swarm to perform scientific exploration	supplied <i>scienceRules</i>	// supplied rules used to investigate the		// types of asteroids to explore	supplied <i>plan</i>	// supplied plan for the subswarm to		// achieve system-wide scientific goals	<i>allocationStrategy</i>	// allocation strategy for the subswarm to		// perform scientific exploration	<i>plan</i>	// plan for the subswarm to achieve		// system-wide scientific goals	<i>scienceRules</i>	// rules used to investigate the types of		// asteroids to explore	<i>subswarmSpacecraft</i>	// vector of the spacecraft in the subswarm	<i>leaderSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the		// subswarm	<i>allocationStratMsg</i>	// newly devised subswarm allocation		// strategy message to be sent out to		// <i>leader</i> agreeing with or disagreeing with		// the newly devised allocation strategy	<i>planMsg</i>	// newly devised subswarm plan to achieve		// system-wide scientific goals	<i>leaderElecVoteMsg</i>	// message to be sent to vote for the ruler		// of the <i>leader</i> spacecraft for a subswarm
<i>leaderSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the																																																		
	// subswarm																																																		
supplied <i>allocationStrategy</i>	// supplied allocation strategy for the sub-																																																		
	// swarm to perform scientific exploration																																																		
supplied <i>scienceRules</i>	// supplied rules used to investigate the																																																		
	// types of asteroids to explore																																																		
supplied <i>plan</i>	// supplied plan for the subswarm to																																																		
	// achieve system-wide scientific goals																																																		
<i>allocationStrategy</i>	// allocation strategy for the subswarm to																																																		
	// perform scientific exploration																																																		
<i>plan</i>	// plan for the subswarm to achieve																																																		
	// system-wide scientific goals																																																		
<i>scienceRules</i>	// rules used to investigate the types of																																																		
	// asteroids to explore																																																		
<i>subswarmSpacecraft</i>	// vector of the spacecraft in the subswarm																																																		
<i>leaderSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the																																																		
	// subswarm																																																		
<i>allocationStratMsg</i>	// newly devised subswarm allocation																																																		
	// strategy message to be sent out to																																																		
	// <i>leader</i> agreeing with or disagreeing with																																																		
	// the newly devised allocation strategy																																																		
<i>planMsg</i>	// newly devised subswarm plan to achieve																																																		
	// system-wide scientific goals																																																		
<i>leaderElecVoteMsg</i>	// message to be sent to vote for the ruler																																																		
	// of the <i>leader</i> spacecraft for a subswarm																																																		
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensure the correctness of the planning done by a <i>leader</i> spacecraft through redundant checking and agreement voting.																																																			
Safety - None.																																																			

Passive Variation Point Schema for the Leader Planner Role

Role Schema: LeaderPlanner	Schema ID: LP- Active
Variation Point: Active	
Inherits: SC-Core, LP-Passive	
Parameters of Variation: P9=True; P10=Active; P11=True; P12=True; P13=True; P14=True	
Requirements: V_L1, V_L2, V_L3, V_L4, V_L5, V_L6, V_L7, V_L8, V_L9, V_L11	
Description: Provides the elements of an active <i>leader</i> spacecraft (i.e., a spacecraft actively in charge) to be able to manage, plan and coordinate the spacecraft of a subswarm.	
Activities and Protocols: CoordinateLeaderElection, DecideDataToGather, PerformSubswarmAlloc, PerformSubswarmPlan, AssignTeams, ComSwitchToActive, DirectWorker, GetConfirmFromPassive, InitLeaderElection, SendNewAllocationStrat, SendNewPlan, RedistributeDuties, VoteLeaderElection	
Permissions:	
Reads -	
<i>allocationStrategy</i>	// current allocation strategy for the sub- // swarm to perform scientific exploration
<i>scienceRules</i>	// current rules used to investigate the // types of asteroids to explore
<i>subswarmSpacecraft</i>	// vector of the spacecraft in the subswarm
<i>subswarmSpacecraftPos</i>	// vector of the all the spacecrafts current // positions in the subswarm
<i>leaderSpacecraft</i>	// vector of the <i>leaders</i> in the subswarm
<i>plan</i>	// current plan for the subswarm to achieve // system-wide scientific goals
<i>currentGoal</i>	// current goal of the spacecraft
Changes -	
<i>allocationStrategy</i>	// allocation strategy for the subswarm to // perform scientific exploration
<i>plan</i>	// plan for the subswarm to achieve // system-wide scientific goals
<i>scienceRules</i>	// rules used to investigate the types of // asteroids to explore
<i>subswarmSpacecraft</i>	// vector of the spacecraft in the subswarm
<i>leaderSpacecraft</i>	// vector of the <i>leaders</i> in the subswarm
Generates -	
<i>newAllocationStratMsg</i>	// newly devised subswarm allocation // strategy message to be sent out to // spacecraft in a subswarm
<i>newPlanMsg</i>	// newly devised subswarm plan to achieve // system-wide scientific goals
<i>leaderElecVoteMsg</i>	// message to be sent to vote for the ruler // of the <i>leader</i> spacecraft for a subswarm
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to optimize the configuration and plans of the subswarm to achieve system-wide goals.	
Safety -	
None.	

Active Variation Point Schema for the Leader Planner Role

Role Variation Points Schema: LeaderKnowledgeLevel		Schemata ID: LKL
Parameters of Variation: P15, P16, P17, P18, P19		
Description:		
Provides the <i>messenger</i> spacecraft with the functionality to be able to facilitate the communication network of the subswarm and the ability to travel to a destination point so that the spacecraft can relay the discovered information back to mission control.		
Variation Points:		
<u>Subswarm:</u>	The core elements of a <i>leader</i> spacecraft to be able to facilitate the management and coordination of its subswarm. [LKL-Subswarm]	
Partial:	The functionality of a <i>leader</i> spacecraft to manage and coordinate several subswarms. [LKL-Partial]	
Full:	The functionality of a <i>leader</i> spacecraft to manage and coordinate the entire swarms. [LKL-Full]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time, however the spacecraft may switch its operating variation point (e.g., from Subswarm to Partial-Swarm) at runtime. All <i>leader</i> spacecraft shall have the Subswarm variation point as a commonality. <i>Leader</i> spacecraft with the Partial-Swarm variation point shall also include all functionality of Subswarm. Likewise, all <i>leader</i> spacecraft with the Partial-Swarm variation point shall have the functionality of the Full-Swarm.		

Leader Knowledge Role Variation Points Schema

Role Schema: LeaderKnowledgeLevel	Schema ID: LKL-Subswarm
Variation Point: Subswarm	
Inherits: SC-Core	
Parameters of Variation: P15=True; P16=True; P17=True; P18=Subswarm; P19=True	
Requirements: V_L10, V_L12, V_L13, V_L14, V_L15, V_L16, V_L17	
Description: Provides the core elements of a <i>leader</i> spacecraft to be able to facilitate the management and coordination of its subswarm.	
Activities and Protocols: CalculatePartModel, OverseeSubSwarmDataFlow, PerformSubswarmReconfig, <u>AcceptSubswarmChangeVelocityBid</u> , <u>MoveNewPositionCom</u> , <u>ReqDataFlow</u> , <u>ReqSubswarmVelocityBids</u> , <u>SendModelPartMessenger</u> , <u>SendSubswarmVelocityBidConfirm</u>	
Permissions:	
Reads -	
<i>subswarmSpacecraft</i>	// vector of the spacecraft in the subswarm
<i>subswarmSpacecraftPos</i>	// vector of the all the spacecrafts current // positions in the subswarm
<i>leaderSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the // subswarm
supplied <i>velocityBidRec</i>	// vector of received change of velocity bid
Changes -	
<i>asteroidModel</i>	// current model of an asteroid to send
<i>subswarmSpacecraft</i>	// vector of the spacecraft in the subswarm
<i>subswarmSpacecraftPos</i>	// vector of the all thespacecrafts positions // in the subswarm
<i>leaderSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the // subswarm
Generates -	
<i>partialAsteroidModel</i>	// derived partial model to send to a // <i>messenger</i> so that spacecraft can avoid // collisions with asteroids
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to optimize the configuration and plans of the subswarm to achieve subswarm goals.	
Safety -	
Avoiding collisions by maintaining and coordinating spacecraft positions and movements.	

Subswarm Knowledge Variation Point Schema for the Leader Knowledge Role

Role Schema: LeaderKnowledgeLevel	Schema ID: LKL-Partial
Variation Point: Parial	
Inherits: LKL=Subswarm	
Parameters of Variation: P15=True; P16=True; P17=True; P18=Partial-Subswarm; P19=True	
Requirements: V_L10, V_L12, V_L13, V_L14, V_L15, V_L16, V_L17	
Description: Provides the functionality of a <i>leader</i> spacecraft to manage and coordinate several subswarms	
Activities and Protocols: CoordinateReconfigs, CoordinateSubswarms, OverseePartSwarmDataFlow, PerformPartSwarmReconfig, <u>AcceptPartSwarmChangeVelocityBid</u> , <u>ComSubswarmsReconfig</u> , <u>MoveNewPositionCom</u> , <u>ReqPartSwarmVelocityBids</u> , <u>SendModelPartMessenger</u> , <u>SendPartswarmVelocityBidConfirm</u>	
Permissions:	
Reads -	
<i>subswarmsPos</i>	// positions of several subswarms to // coordinate
<i>subswarmsLeaders</i>	// vector of the <i>leader</i> spacecraft in charge // of different subswarms
Changes -	
<i>subswarmsPos</i>	// positions of several subswarms to // coordinate
Generates -	
<i>subswarmReconfigMsg</i>	// message to be sent to the <i>leaders</i> of // multiple subswarms to reconfigure
Responsibilities:	
Liveness -	If the spacecraft is functioning properly, this role will eventually be able to optimize the configuration and plans of several subswarms to achieve system-wide goals.
Safety -	Avoiding collisions by maintaining and coordinating spacecraft positions and movements of several subswarm's spacecraft.

Partial-Swarm Knowledge Variation Point Schema for the Leader Knowledge Role

Role Schema: LeaderKnowledgeLevel	Schema ID: LKL-Full
Variation Point: Full	
Inherits: LKL-Partial	
Parameters of Variation: P15=True; P16=True; P17=True; P18=Full-Swarm; P19=True	
Requirements: V_L10, V_L12, V_L13, V_L14, V_L15, V_L16, V_L17	
Description: Provides the functionality of a <i>leader</i> spacecraft to manage and coordinate the entire swarms	
Activities and Protocols: CoordinateReconfigs, CoordinateSwarm, ManageSwarm OverseeSwarmDataFlow, PerformSwarmReconfig, <u>AcceptSwarmChangeVelocityBid</u> , <u>ComSwarmsReconfig</u> , <u>MoveNewPositionCom</u> , <u>ReqPartSwarmVelocityBids</u> , <u>SendSwarmVelocityBidConfirm</u>	
Permissions:	
Reads -	
<i>subswarmsPos</i>	// positions of several subswarms to // coordinate
<i>subswarmsLeaders</i>	// vector of the <i>leader</i> spacecraft in charge // of different subswarms
supplied <i>subswarmGoals</i>	// current scientific goals of all the // subswarms
Changes -	
<i>subswarmsPos</i>	// positions of all subswarms to coordinate
Generates -	
<i>swarmNotifyMsg</i>	// message to be sent to all <i>messengers</i> of // the swarm
<i>swarmReconfigMsg</i>	// message to be sent to the <i>leaders</i> of // all subswarms to reconfigure
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to optimize the configuration and plans of the swarm to achieve system-wide goals.	
Safety -	
Avoiding collisions by maintaining and coordinating spacecraft positions and movements of all spacecraft in the swarm.	

Full-Swarm Knowledge Variation Point Schema for the Leader Knowledge Role

Role Variation Points Schema: Worker		Schemata ID: W
Parameters of Variation: P26, P28, P30, P31, P32, P33, P34, P35, P36, P37, P38, P39, P40, P41, P42, P43, P44, P45, P46, P47, P48		
Description:		
This role and its variation points comprise the specialized functionality for the <i>worker</i> spacecraft of the swarm and each of the specialized instrumentations of the <i>worker</i> spacecraft.		
Variation Points:		
<u>Core:</u>	The core elements of a <i>worker</i> spacecraft to be able to explore the asteroid belt and pursue and satisfy scientific exploration goals. [W-Core]	
Imager:	The functionality required of those <i>worker</i> spacecraft equipped with a visible imager to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-Imager]	
NIRSpec:	The functionality required of those <i>worker</i> spacecraft equipped with a near-infrared spectrometer to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-NIRSpec]	
XRayspec:	The functionality required of those <i>worker</i> spacecraft equipped with an X-ray spectrometer to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W- XRayspec]	
GRaySpec:	The functionality required of those <i>worker</i> spacecraft equipped with a Gamma-ray spectrometer to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-GRaySpec]	
NeuSpec:	The functionality required of those <i>worker</i> spacecraft equipped with a Neutron spectrometer to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-NeuSpec]	
Altimeter:	The functionality required of those <i>worker</i> spacecraft equipped with an altimeter to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-Altimeter]	
Magneto:	The functionality required of those <i>worker</i> spacecraft equipped with a radio science/magnetometer to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-Magneto]	
Radiometer:	The functionality required of those <i>worker</i> spacecraft equipped with a radio sounder/infrared radiometer to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-Radiometer]	
NMSpec:	The functionality required of those <i>worker</i> spacecraft equipped with a neutral mass spectrometer to pursue and satisfy scientific exploration goals specific to its specialized onboard equipment. [W-NMSpec]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time.		

Worker Role Variation Point Schema

Role Schema: Worker	Schema ID: W-Core
Variation Point: Core	
Inherits: None	
Parameters of Variation: N/A	
Requirements: V_W2, V_W3, V_W4,	
Description: Provides the <i>worker</i> spacecraft with the functionality to be able to explore the asteroid belt and pursue and satisfy scientific exploration goals.	
Activities and Protocols: <u>CheckStatus</u> , <u>CollectPrelimAsteroidData</u> , <u>EvaluateOpportunity</u> , <u>EvaluateScienceGoals</u> , <u>AcceptAsteroidData</u> , <u>SendArchiveData</u> , <u>SendAsteroidData</u> , <u>SendPrelimAsteroidData</u>	
Permissions:	
Reads -	
<i>asteroidID</i>	// identification of the current asteroid // under exploration
<i>asteriodData</i>	// current collected data of an asteroid
<i>currentGoal</i>	// current goal of the spacecraft
<i>workerSpacecraft</i>	// vector of the <i>workers</i> in the subswarm
<i>likeWorkerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm with the same specialized // instrumentation
<i>systemStatus</i>	// current status of the spacecraft
<i>riskForSystemFactor</i>	// current risk to spacecraft
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>curScienceGoalFactor</i>	// current spacecraft scientific goal factor
<i>subswarmVector</i>	// vector of nearby spacecraft
Changes -	
<i>asteriodData</i>	// collected data of an asteroid
<i>asteroidID</i>	// identification of the current asteroid // under exploration
<i>currentGoal</i>	// current goal of the spacecraft
<i>systemStatus</i>	// current status of the spacecraft
Generates -	
<i>asteriodDataMsg</i>	// message to be sent containing the // current collected data of an asteroid
<i>prelimAsteriodDataMsg</i>	// message to be sent to a <i>leader</i> contain- // ing preliminary data of an asteroid
<i>prelimAsteroidData</i>	// collected preliminary data of an asteroid
<i>scienceGoalsEval</i>	// spacecraft's evaluation of its current // science goals and the advantage of the // opportunity to explore an asteroid for // preliminary data
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to optimize the scientific exploration of the swarm by taking advantage of opportunities for scientific exploration when they are presented.	
Safety -	
None.	

Core Variation Point Schema for the Worker Role

Role Schema: Worker	Schema ID: W-Imager
Variation Point: Imager	
Inherits: W-Core	
Parameters of Variation: P30=Narrow-Scope OR Wide-Scope; P31=True; P32=True; P33=True; P34=True; P35=True	
Requirements: V_W6, V_W7, V_W8, V_W9, V_W10, V_W11	
Description: Provides the <i>worker</i> spacecraft with the functionality to be able to utilize its onboard visible imager to pursue and satisfy scientific exploration goals.	
Activities and Protocols: CalculateAsteroidLocation, CheckMemoryStatus, Construct3DModel, DetectAsteroid, GatherImagingData, GatherModelData, GatherPhotogeologyData, GenerateRoughModel, PerformPhotogeolgy, TakeAsteroidImages, <u>Accept3DModel</u> , <u>CollaborateOn3DModel</u> , <u>Send3DModel</u> , <u>SendAsteroidLocation</u> , <u>SendAsteroidImages</u> , <u>SendPhotogeologyData</u> , <u>SendRoughModel</u>	
Permissions: Reads - <ul style="list-style-type: none"> <i>altimeterSpecSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft // equipped with an altimeter <i>imagerSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft // equipped with a visible imager <i>magnetoSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft // equipped with a visible imager <i>memoryStatus</i> // status of the spacecraft's memory to // ensure sufficient space for data <i>workerSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft supplied <i>asteroid3DModel</i> // 3D model of an asteroid based on the // collected data Changes - <ul style="list-style-type: none"> <i>asteroid3DModel</i> // 3D model of an asteroid based on the // collected data <i>memoryStatus</i> // status of the spacecraft's memory to // ensure sufficient space for data Generates - <ul style="list-style-type: none"> <i>asteroid3DModel</i> // 3D model of an asteroid based on the // collected data <i>asteroidData</i> // asteroid data collected <i>asteroidImages</i> // vector of images taken of an asteroid <i>asteroidLocation</i> // derived location of an asteroid <i>asteroidRoughModel</i> // a rough model of the asteroid to send to // other asteroid for navigation purposes <i>photogeologyData</i> // photogeology data generated by the // spacecraft of an asteroid 	
Responsibilities: Liveness - <ul style="list-style-type: none"> If the spacecraft is functioning properly, this role will eventually be able to ensuring the satisfaction of science goals pertaining to the visible imager onboard instrumentation. Safety - <ul style="list-style-type: none"> Accuracy of the asteroid model to be sent to other spacecraft to prevent spacecraft-asteroid collisions. 	

Visible Imager Variation Point Schema for the Worker Role

Role Schema: Worker	Schema ID: W-GRaySpec
Variation Point: GRaySpec	
Inherits: W-Core	
Parameters of Variation: P38=True	
Requirements: V_W14	
Description: Provides the <i>worker</i> spacecraft with the functionality to be able to utilize its onboard Gamma-ray spectrometer to pursue and satisfy scientific exploration goals related to detecting an asteroid's heavy element abundance mapping.	
Activities and Protocols: CalculateHeavyEleMapping, CheckDetectedElements, DetectHeavyEleAbundance, PerformGammaRaySpectrometry, SendHeavyEleMapping	
Permissions: Reads - <i>asteroidID</i> // identification of the current asteroid // under exploration <i>heavyElementList</i> // list of the characterization of the heavy // elements to detect <i>gammaRaySpecSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft // equipped with an Gamma-ray // spectrometer <i>workerSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft Changes - <i>asteroidData</i> // data collected by the spacecraft of an // asteroid Generates - <i>asteroidMajorEleMap</i> // calculated heavy element abundance // mapping of an asteroid	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensuring the satisfaction of science goals pertaining to the Gamma-ray spectrometer onboard instrumentation. Safety - None.	

Gamma-Ray Spectrometer Variation Point Schema for the Worker Role

Role Schema: Worker	Schema ID: W-Altimeter																																				
Variation Point: Altimeter																																					
Inherits: W-Core																																					
Parameters of Variation: P40=True; P41=True; P42=True; P43=True;																																					
Requirements: V_W16, V_W17, V_W18, V_W19																																					
Description: Provides the <i>worker</i> spacecraft with the functionality to be able to utilize its onboard altimeter to pursue and satisfy scientific exploration goals related to detecting an asteroid's topographic and geomorphic characteristics.																																					
Activities and Protocols: CalculateAsteroidShape, CalculateAsteroidTopography, Construct3DModel, DetectAsteroidShape, GatherGeomorphData, GatherModelData, GatherTopographyData, <u>Accept3DModel</u> , <u>CollaborateOn3DModel</u> , <u>Send3DModel</u> , <u>SendAsteroidShapeData</u> , <u>SendAsteroidTopography</u>																																					
Permissions: Reads - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>asteroidID</i></td> <td style="padding-left: 20px;">// identification of the current asteroid</td> </tr> <tr> <td style="padding-left: 20px;"><i>altimeterSpecSpacecraft</i></td> <td style="padding-left: 20px;">// vector of other nearby <i>worker</i> spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>imagerSpacecraft</i></td> <td style="padding-left: 20px;">// equipped with an altimeter spectrometer</td> </tr> <tr> <td style="padding-left: 20px;"><i>workerSpacecraft</i></td> <td style="padding-left: 20px;">// vector of other nearby <i>worker</i> spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>supplied asteroid3DModel</i></td> <td style="padding-left: 20px;">// 3D model of an asteroid based on the</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// collected data</td> </tr> </table> Changes - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>asteroid3DModel</i></td> <td style="padding-left: 20px;">// 3D model of an asteroid based on the</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// collected data</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroidData</i></td> <td style="padding-left: 20px;">// data collected by the spacecraft of an</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// asteroid</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroidShapeData</i></td> <td style="padding-left: 20px;">// calculated shape data of an asteroid</td> </tr> </table> Generates - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>asteroid3DModel</i></td> <td style="padding-left: 20px;">// 3D model of an asteroid based on the</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// collected data</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroidShapeData</i></td> <td style="padding-left: 20px;">// calculated shape data of an asteroid</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroidTopographyData</i></td> <td style="padding-left: 20px;">// calculated topography data of an</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// asteroid</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroidGeomorphData</i></td> <td style="padding-left: 20px;">// calculated geomorphology data of an</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">// asteroid</td> </tr> </table>		<i>asteroidID</i>	// identification of the current asteroid	<i>altimeterSpecSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft	<i>imagerSpacecraft</i>	// equipped with an altimeter spectrometer	<i>workerSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft	<i>supplied asteroid3DModel</i>	// 3D model of an asteroid based on the		// collected data	<i>asteroid3DModel</i>	// 3D model of an asteroid based on the		// collected data	<i>asteroidData</i>	// data collected by the spacecraft of an		// asteroid	<i>asteroidShapeData</i>	// calculated shape data of an asteroid	<i>asteroid3DModel</i>	// 3D model of an asteroid based on the		// collected data	<i>asteroidShapeData</i>	// calculated shape data of an asteroid	<i>asteroidTopographyData</i>	// calculated topography data of an		// asteroid	<i>asteroidGeomorphData</i>	// calculated geomorphology data of an		// asteroid
<i>asteroidID</i>	// identification of the current asteroid																																				
<i>altimeterSpecSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft																																				
<i>imagerSpacecraft</i>	// equipped with an altimeter spectrometer																																				
<i>workerSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft																																				
<i>supplied asteroid3DModel</i>	// 3D model of an asteroid based on the																																				
	// collected data																																				
<i>asteroid3DModel</i>	// 3D model of an asteroid based on the																																				
	// collected data																																				
<i>asteroidData</i>	// data collected by the spacecraft of an																																				
	// asteroid																																				
<i>asteroidShapeData</i>	// calculated shape data of an asteroid																																				
<i>asteroid3DModel</i>	// 3D model of an asteroid based on the																																				
	// collected data																																				
<i>asteroidShapeData</i>	// calculated shape data of an asteroid																																				
<i>asteroidTopographyData</i>	// calculated topography data of an																																				
	// asteroid																																				
<i>asteroidGeomorphData</i>	// calculated geomorphology data of an																																				
	// asteroid																																				
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensuring the satisfaction of science goals pertaining to the altimeter onboard instrumentation.																																					
Safety - Accuracy of the asteroid model to be sent to other spacecraft to prevent spacecraft-asteroid collisions.																																					

Altimeter Variation Point Schema for the Worker Role

Role Schema: Worker	Schema ID: W-Magneto																				
Variation Point: Magneto																					
Inherits: W-Core																					
Parameters of Variation: P44=True; P45=True; P46=True																					
Requirements: V_W20, V_W21, V_W22 V_W23																					
Description: Provides the <i>worker</i> spacecraft with the functionality to be able to utilize its onboard radio science/magnetometer instrumentation to pursue and satisfy scientific exploration goals related to detecting an asteroid's gravity and magnetic fields.																					
Activities and Protocols: CalculateInteriorMakeup, DetectGravityField, DetectMagneticField, GatherModelData, MeasureAsteroidInterior, MeasureGravityField, MeasureMagneticField, CollaborateOn3DModel, Send3DModel, SendGravityFieldData, SendMagneticFieldData																					
Permissions: Reads - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>asteroidID</i></td> <td style="padding-left: 20px;">// identification of the current asteroid</td> </tr> <tr> <td style="padding-left: 20px;"><i>magnetoSpacecraft</i></td> <td style="padding-left: 20px;">// vector of other nearby <i>worker</i> spacecraft // equipped with a radio // science/magnetometer</td> </tr> <tr> <td style="padding-left: 20px;"><i>altimeterSpecSpacecraft</i></td> <td style="padding-left: 20px;">// vector of other nearby <i>worker</i> spacecraft // equipped with an altimeter spectrometer</td> </tr> <tr> <td style="padding-left: 20px;"><i>imagerSpacecraft</i></td> <td style="padding-left: 20px;">// vector of other nearby <i>worker</i> spacecraft // equipped with a visible imager</td> </tr> <tr> <td style="padding-left: 20px;"><i>workerSpacecraft</i> supplied <i>asteroid3DModel</i></td> <td style="padding-left: 20px;">// vector of other nearby <i>worker</i> spacecraft // 3D model of an asteroid based on the // collected data</td> </tr> </table> Changes - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>asteroidData</i></td> <td style="padding-left: 20px;">// data collected by the spacecraft of an // asteroid</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroid3DModel</i></td> <td style="padding-left: 20px;">// 3D model of an asteroid based on the // collected data</td> </tr> </table> Generates - <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;"><i>asteroid3DModel</i></td> <td style="padding-left: 20px;">// 3D model of an asteroid based on the // collected data</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroidGravityFieldData</i></td> <td style="padding-left: 20px;">// calculated gravity field data of an // asteroid</td> </tr> <tr> <td style="padding-left: 20px;"><i>asteroidMagneticFieldData</i></td> <td style="padding-left: 20px;">// calculated magnetic field data of an // asteroid</td> </tr> </table>		<i>asteroidID</i>	// identification of the current asteroid	<i>magnetoSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft // equipped with a radio // science/magnetometer	<i>altimeterSpecSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft // equipped with an altimeter spectrometer	<i>imagerSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft // equipped with a visible imager	<i>workerSpacecraft</i> supplied <i>asteroid3DModel</i>	// vector of other nearby <i>worker</i> spacecraft // 3D model of an asteroid based on the // collected data	<i>asteroidData</i>	// data collected by the spacecraft of an // asteroid	<i>asteroid3DModel</i>	// 3D model of an asteroid based on the // collected data	<i>asteroid3DModel</i>	// 3D model of an asteroid based on the // collected data	<i>asteroidGravityFieldData</i>	// calculated gravity field data of an // asteroid	<i>asteroidMagneticFieldData</i>	// calculated magnetic field data of an // asteroid
<i>asteroidID</i>	// identification of the current asteroid																				
<i>magnetoSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft // equipped with a radio // science/magnetometer																				
<i>altimeterSpecSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft // equipped with an altimeter spectrometer																				
<i>imagerSpacecraft</i>	// vector of other nearby <i>worker</i> spacecraft // equipped with a visible imager																				
<i>workerSpacecraft</i> supplied <i>asteroid3DModel</i>	// vector of other nearby <i>worker</i> spacecraft // 3D model of an asteroid based on the // collected data																				
<i>asteroidData</i>	// data collected by the spacecraft of an // asteroid																				
<i>asteroid3DModel</i>	// 3D model of an asteroid based on the // collected data																				
<i>asteroid3DModel</i>	// 3D model of an asteroid based on the // collected data																				
<i>asteroidGravityFieldData</i>	// calculated gravity field data of an // asteroid																				
<i>asteroidMagneticFieldData</i>	// calculated magnetic field data of an // asteroid																				
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensuring the satisfaction of science goals pertaining to the radio science/magnetometer onboard instrumentation. Safety - Accuracy of the asteroid model to be sent to other spacecraft to prevent spacecraft-asteroid collisions.																					

Radio Science/Magnetometer Variation Point Schema for the Worker Role

Role Schema: Worker	Schema ID: W-NMSpec
Variation Point: NmSpec	
Inherits: W-Core	
Parameters of Variation: P48=True	
Requirements: V_W25	
Description: Provides the <i>worker</i> spacecraft with the functionality to be able to utilize its onboard neutral mass spectrometer to pursue and satisfy scientific exploration goals related to detecting an asteroid's volatile characterization.	
Activities and Protocols: CalculateVolatileCharacterization, CheckDetectedElements, DetectVolatileEleAbundance, PerformNeutralMassSpectrometry, SendVolatileCharacterization	
Permissions: Reads - <i>asteroidID</i> // identification of the current asteroid // under exploration <i>volatileElementList</i> // list of the characterization of the volatile // elements to detect <i>neutronSpecSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft // equipped with a Neutron // spectrometer <i>workerSpacecraft</i> // vector of other nearby <i>worker</i> spacecraft Changes - <i>asteroidData</i> // data collected by the spacecraft of an // asteroid Generates - <i>volatileCharacterization</i> // calculated volatile characterization of an // asteroid <i>volatileEleMap</i> // calculated volatile element abundance // mapping of an asteroid	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensuring the satisfaction of science goals pertaining to the neutral mass spectrometer onboard instrumentation. Safety - None.	

Neutral Mass Spectrometer Variation Point Schema for the Worker Role

Role Variation Points Schema: WorkerCooperationLevel		Schemata ID: WCL
Parameters of Variation: P29		
Description:		
Provides the <i>worker</i> spacecraft with the functionality to be able to either work within the context of a subswarm to collect data on a targeted asteroid or the ability to work as in individual to gather scientific data on an asteroid without the need to collaborate with other <i>worker</i> spacecraft.		
Variation Points:		
WorkInTeam:	The functionality of a <i>worker</i> spacecraft to work within the context of a subswarm to form virtual instruments to achieve scientific goals. [WCL-Team]	
WorkSolo:	The functionality of a <i>worker</i> spacecraft to work as an individual to achieve scientific goals. Particularly used to gather preliminary data on an asteroid so that a <i>leader</i> spacecraft can then decide whether an asteroid is interesting enough to send an entire subswarm to explore it in detail. [WCL-Solo]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time.		

Worker Cooperation Level Role Variation Point Schema

Role Schema: WorkerCooperationLevel	Schema ID: WIS-Team
Variation Point: WorkInTeam	
Inherits: W-Core	
Parameters of Variation: P29=False	
Requirements: V_W5	
Description: Provides the functionality of a <i>worker</i> spacecraft to work within the context of a subswarm to form virtual instruments to achieve scientific goals	
Activities and Protocols: DecideNewGoal, CheckGoalStatus, MoveToJoinSubswarm, <u>AcceptJoinSubswarm</u> , <u>NotifySubswarmLeader</u> , <u>RejectJoinSubswarm</u> , <u>RequestNewGoal</u>	
Permissions:	
Reads -	
<i>asteroidID</i>	// identification of the current asteroid // under exploration
<i>asteriodData</i>	// current collected data of an asteroid
<i>currentGoal</i>	// current goal of the spacecraft
<i>workerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm
<i>messengerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>likeWorkerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm with the same specialized // instrumentation
<i>subswarmID</i>	// identification of the subswarm it belongs
<i>systemStatus</i>	// current status of the spacecraft
Changes -	
<i>asteriodData</i>	// collected data of an asteroid
<i>asteroidID</i>	// identification of the current asteroid // under exploration
<i>currentGoal</i>	// current goal of the spacecraft
<i>systemStatus</i>	// current status of the spacecraft
Generates -	
<i>scienceGoalsEval</i>	// spacecraft's evaluation of its current // science goals and the advantage of the // opportunity to explore an asteroid for // preliminary data
Responsibilities:	
Liveness - If the spacecraft is functioning properly, this role will eventually be able to achieve the scientific goals of the subswarm.	
Safety - None.	

**Work in a Subswarm Variation Point Schema for the Worker Cooperation Level
Role**

Role Schema: WorkerCooperationLevel	Schema ID: WIS-Solo																														
Variation Point: WorkSolo																															
Inherits: W-Core																															
Parameters of Variation: P29=True																															
Requirements: V_W5																															
<p>Description: Provides the functionality of a <i>worker</i> spacecraft to work as an individual to achieve scientific goals. Particularly used to gather preliminary data on an asteroid so that a <i>leader</i> spacecraft can then decide whether an asteroid is interesting enough to send an entire subswarm to explore it in detail.</p>																															
<p>Activities and Protocols: DecideNewGoal, EvaluateCurrentAsteroid, MoveToNewAsteroid, SendPrelimAsteroidData</p>																															
<p>Permissions:</p> <p>Reads -</p> <table> <tr> <td><i>asteroidID</i></td> <td>// identification of the current asteroid // under exploration</td> </tr> <tr> <td><i>asteriodData</i></td> <td>// current collected data of an asteroid</td> </tr> <tr> <td><i>currentGoal</i></td> <td>// current goal of the spacecraft</td> </tr> <tr> <td><i>messengerSpacecraft</i></td> <td>// vector of the <i>worker</i> spacecraft in the // subswarm</td> </tr> <tr> <td><i>messengerSpacecraft</i></td> <td>// vector of the <i>leader</i> spacecraft in the // subswarm</td> </tr> <tr> <td><i>position</i></td> <td>// current spacecraft position</td> </tr> <tr> <td><i>velocityIncrement</i></td> <td>// current spacecraft velocity increment</td> </tr> <tr> <td><i>likeWorkerSpacecraft</i></td> <td>// vector of the <i>worker</i> spacecraft in the // subswarm with the same specialized // instrumentation</td> </tr> <tr> <td><i>subswarmID</i></td> <td>// identification of the subswarm it belongs</td> </tr> <tr> <td><i>systemStatus</i></td> <td>// current status of the spacecraft</td> </tr> </table> <p>Changes -</p> <table> <tr> <td><i>prelimAsteroidData</i></td> <td>// collected preliminary data of an asteroid</td> </tr> <tr> <td><i>asteriodData</i></td> <td>// current collected data of an asteroid</td> </tr> <tr> <td><i>currentGoal</i></td> <td>// current goal of the spacecraft</td> </tr> </table> <p>Generates -</p> <table> <tr> <td><i>prelimAsteriodDataMsg</i></td> <td>// message to be sent to a <i>leader</i> // containing preliminary data of an // asteroid</td> </tr> <tr> <td><i>prelimAsteroidData</i></td> <td>// collected preliminary data of an asteroid</td> </tr> </table>		<i>asteroidID</i>	// identification of the current asteroid // under exploration	<i>asteriodData</i>	// current collected data of an asteroid	<i>currentGoal</i>	// current goal of the spacecraft	<i>messengerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm	<i>messengerSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the // subswarm	<i>position</i>	// current spacecraft position	<i>velocityIncrement</i>	// current spacecraft velocity increment	<i>likeWorkerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm with the same specialized // instrumentation	<i>subswarmID</i>	// identification of the subswarm it belongs	<i>systemStatus</i>	// current status of the spacecraft	<i>prelimAsteroidData</i>	// collected preliminary data of an asteroid	<i>asteriodData</i>	// current collected data of an asteroid	<i>currentGoal</i>	// current goal of the spacecraft	<i>prelimAsteriodDataMsg</i>	// message to be sent to a <i>leader</i> // containing preliminary data of an // asteroid	<i>prelimAsteroidData</i>	// collected preliminary data of an asteroid
<i>asteroidID</i>	// identification of the current asteroid // under exploration																														
<i>asteriodData</i>	// current collected data of an asteroid																														
<i>currentGoal</i>	// current goal of the spacecraft																														
<i>messengerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm																														
<i>messengerSpacecraft</i>	// vector of the <i>leader</i> spacecraft in the // subswarm																														
<i>position</i>	// current spacecraft position																														
<i>velocityIncrement</i>	// current spacecraft velocity increment																														
<i>likeWorkerSpacecraft</i>	// vector of the <i>worker</i> spacecraft in the // subswarm with the same specialized // instrumentation																														
<i>subswarmID</i>	// identification of the subswarm it belongs																														
<i>systemStatus</i>	// current status of the spacecraft																														
<i>prelimAsteroidData</i>	// collected preliminary data of an asteroid																														
<i>asteriodData</i>	// current collected data of an asteroid																														
<i>currentGoal</i>	// current goal of the spacecraft																														
<i>prelimAsteriodDataMsg</i>	// message to be sent to a <i>leader</i> // containing preliminary data of an // asteroid																														
<i>prelimAsteroidData</i>	// collected preliminary data of an asteroid																														
<p>Responsibilities:</p> <p>Liveness - If the spacecraft is functioning properly, this role will eventually be able to achieve its own scientific goals and communicate the findings to a <i>leader</i> spacecraft.</p> <p>Safety - None.</p>																															

Work Individually Variation Point Schema for the Worker Cooperation Level Role

Role Variation Points Schema: WorkerImagerScope		Schemata ID: WIS
Parameters of Variation: P30		
Description:		
Provides the <i>messenger</i> spacecraft with the functionality to be able to facilitate the communication network of the subswarm and the ability to travel to a destination point so that the spacecraft can relay the discovered information back to mission control.		
Variation Points:		
Wide-Scope:	The functionality of a <i>worker</i> spacecraft with a visible imager to operate its wide-scope visible imager onboard instrumentation. [WIS-Wide]	
Narrow-Scope:	The functionality of a <i>worker</i> spacecraft with a visible imager to operate its narrow-scope visible imager onboard instrumentation. [WIS-Narrow]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time.		

Worker Visible Imager Scope Role Variation Point Schema

Role Schema: WorkerImagerScope	Schema ID: WIS-Wide
Variation Point: Wide-Scope	
Inherits: W-Imager	
Parameters of Variation: P30=Wide-Scope	
Requirements: V_W6	
Description: Provides a <i>worker</i> spacecraft with a visible imager to operate its wide-scope visible imager onboard instrumentation to perform scientific exploration and satisfy its scientific goals.	
Activities and Protocols: AnalyzeWideImage, CheckImagerScope, FocusWideImager, IdentifyTargetAsteroids, TakeWideScopeImage, WideImagerScopeShutdown, NotifyWideImagerFailure, SendWideImagerToLeader	
Permissions: Reads - <i>imagerSpacecraft</i> // vector of all other imager <i>worker</i> spacecraft in the subswarm <i>imagerStatus</i> // current status of the wide-scoped imager <i>wideImagerSpacecraft</i> // vector of all other wide-scoped imager <i>worker</i> spacecraft in the subswarm <i>wideImagerFocusVal</i> // value of the focus of the imager Changes - <i>asteroidData</i> // collected and derived data of an asteroid <i>imagerStatus</i> // current status of the wide-scoped imager <i>wideImagerFocusVal</i> // value of the focus of the imager Generates - <i>asteroidImageData</i> // data generated from the analysis of an image taken with a wide-scoped visible imager <i>wideImagerFailureMsg</i> // message to be sent to other wide-scoped imager <i>worker</i> spacecraft notifying of a hardware failure	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensuring the satisfaction of science goals pertaining to the visible imager onboard instrumentation. Safety - None.	

Wide-Scope Imager Variation Point Schema for the Worker Visible Imager Role

Role Schema: WorkerImagerScope	Schema ID: WIS-Narrow
Variation Point: Narrow-Scope	
Inherits: W-Imager	
Parameters of Variation: P30=Narrow-Scope	
Requirements: V_W6	
Description: Provides a <i>worker</i> spacecraft with a visible imager to operate its narrow-scope visible imager onboard instrumentation to perform scientific exploration and satisfy its scientific goals.	
Activities and Protocols: AnalyzeNarrowImage, CheckImagerScope, FocusNarrowImager, GetAsteroidDetailsFromImage, NarrowImagerScopeShutdown, TakeNarrowScopeImage, <u>NotifyNarrowImagerFailure</u>	
Permissions: Reads - <ul style="list-style-type: none"> <i>imagerSpacecraft</i> // vector of all other imager <i>worker</i> spacecraft in the subswarm <i>imagerStatus</i> // current status of the narrow-scoped imager <i>narrowImagerSpacecraft</i> // vector of all other narrow-scoped imager <i>worker</i> spacecraft in the subswarm <i>widelImagerFocusVal</i> // value of the focus of the imager Changes - <ul style="list-style-type: none"> <i>asteroidData</i> // collected and derived data of an asteroid <i>imagerStatus</i> // current status of the narrow-scoped imager <i>narrowImagerFocusVal</i> // value of the focus of the imager Generates - <ul style="list-style-type: none"> <i>asteroidImageData</i> // data generated from the analysis of an image taken with a narrow-scoped visible imager <i>narrowImagerFailureMsg</i> // message to be sent to other narrow-scoped imager <i>worker</i> spacecraft notifying of a hardware failure 	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensuring the satisfaction of science goals pertaining to the visible imager onboard instrumentation. Safety - None.	

Narrow-Scope Imager Variation Point Schema for the Worker Visible Imager Role

Role Variation Points Schema: Messenger		Schemata ID: M
Parameters of Variation: P20, P21, P22, P23, P24, P25		
Description:		
Provides the <i>messenger</i> spacecraft with the functionality to be able to facilitate the communication network of the subswarm and the ability to travel to a destination point so that the spacecraft can relay the discovered information back to mission control.		
Variation Points:		
<u>Core:</u>	The core elements of a <i>messenger</i> spacecraft to be able to facilitate the communication network of the swarm. [M-Core]	
LagrangeTravel:	The functionality of a <i>messenger</i> spacecraft to additionally be able to travel to a Lagrange point and communicate the results of asteroid exploration to mission control and other spacecraft. [M-LagrangeTravel]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time.		

Messenger Role Variation Points Schema

Role Schema: Messenger	Schema ID: M-Core																																																
Variation Point: Core																																																	
Inherits: None																																																	
Parameters of Variation: P20=True; P22=dc; P23=True; P25=True																																																	
Requirements: V_M1, V_M3, V_M4, V_M5, V_M7, V_M8																																																	
Description: Provides the spacecraft with the functionality to facilitate the communication network of the subswarm by relaying and coordinating the messages and data discovered through scientific exploration.																																																	
Activities and Protocols: ArchiveData, CheckMemoryStatus, MaintainSpacecraftPosData, AcceptArchiveData, AcceptAsteroidModel, AcceptLeaderMsg, AcceptMessengerMsg, AcceptWorkerMsg, CoordinateMessagesLeader, CoordinateMessageWorker, RelayAsteroidModel, RelayMessageToLeader, RelayMessageToWorker,																																																	
Permissions: Reads - <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;"><i>leaderSpacecraft</i></td> <td style="padding-left: 20px;">// vector of the <i>leaders</i> in the subswarm</td> </tr> <tr> <td style="padding-left: 20px;"><i>messengerSpacecraft</i></td> <td style="padding-left: 20px;">// vector of the <i>workers</i> in the subswarm</td> </tr> <tr> <td style="padding-left: 20px;"><i>memoryStatus</i></td> <td style="padding-left: 20px;">// current status (i.e., functioning or mal-functioning) of the spacecraft's memory</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// system</td> </tr> <tr> <td style="padding-left: 20px;"><i>systemStatus</i></td> <td style="padding-left: 20px;">// current status of the spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>spacecraftOrbitData</i></td> <td style="padding-left: 20px;">// vector of the orbital insertion data of all</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// subswarm spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>spacecraftTrajectoryData</i></td> <td style="padding-left: 20px;">// vector of the trajectory data of all</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// subswarm spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>workerSpacecraft</i></td> <td style="padding-left: 20px;">// vector of the <i>workers</i> in the subswarm</td> </tr> <tr> <td style="padding-left: 20px;"><i>supplied fromSpacecraftID</i></td> <td style="padding-left: 20px;">// spacecraft identification number that</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// sent a message to be relayed</td> </tr> <tr> <td style="padding-left: 20px;"><i>supplied toSpacecraftID</i></td> <td style="padding-left: 20px;">// spacecraft identification number that</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// is to received a relayed message</td> </tr> <tr> <td style="padding-left: 20px;"><i>supplied messageRec</i></td> <td style="padding-left: 20px;">// the message received to be relayed</td> </tr> </table> Changes - <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;"><i>messageRec</i></td> <td style="padding-left: 20px;">// the message received to be relayed</td> </tr> <tr> <td style="padding-left: 20px;"><i>messageHistory</i></td> <td style="padding-left: 20px;">// history log of the messages sent</td> </tr> <tr> <td style="padding-left: 20px;"><i>memoryStatus</i></td> <td style="padding-left: 20px;">// current status (i.e., functioning or mal-functioning) of the spacecraft's memory</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// subswarm spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>spacecraftOrbitData</i></td> <td style="padding-left: 20px;">// vector of the orbital insertion data of all</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// subswarm spacecraft</td> </tr> <tr> <td style="padding-left: 20px;"><i>spacecraftTrajectoryData</i></td> <td style="padding-left: 20px;">// vector of the trajectory data of all</td> </tr> <tr> <td style="padding-left: 20px;"></td> <td style="padding-left: 20px;">// subswarm spacecraft</td> </tr> </table> Generates - <table style="width: 100%; border: none;"> <tr> <td style="padding-left: 20px;"><i>messageHistory</i></td> <td style="padding-left: 20px;">// history log of the messages sent</td> </tr> </table>		<i>leaderSpacecraft</i>	// vector of the <i>leaders</i> in the subswarm	<i>messengerSpacecraft</i>	// vector of the <i>workers</i> in the subswarm	<i>memoryStatus</i>	// current status (i.e., functioning or mal-functioning) of the spacecraft's memory		// system	<i>systemStatus</i>	// current status of the spacecraft	<i>spacecraftOrbitData</i>	// vector of the orbital insertion data of all		// subswarm spacecraft	<i>spacecraftTrajectoryData</i>	// vector of the trajectory data of all		// subswarm spacecraft	<i>workerSpacecraft</i>	// vector of the <i>workers</i> in the subswarm	<i>supplied fromSpacecraftID</i>	// spacecraft identification number that		// sent a message to be relayed	<i>supplied toSpacecraftID</i>	// spacecraft identification number that		// is to received a relayed message	<i>supplied messageRec</i>	// the message received to be relayed	<i>messageRec</i>	// the message received to be relayed	<i>messageHistory</i>	// history log of the messages sent	<i>memoryStatus</i>	// current status (i.e., functioning or mal-functioning) of the spacecraft's memory		// subswarm spacecraft	<i>spacecraftOrbitData</i>	// vector of the orbital insertion data of all		// subswarm spacecraft	<i>spacecraftTrajectoryData</i>	// vector of the trajectory data of all		// subswarm spacecraft	<i>messageHistory</i>	// history log of the messages sent
<i>leaderSpacecraft</i>	// vector of the <i>leaders</i> in the subswarm																																																
<i>messengerSpacecraft</i>	// vector of the <i>workers</i> in the subswarm																																																
<i>memoryStatus</i>	// current status (i.e., functioning or mal-functioning) of the spacecraft's memory																																																
	// system																																																
<i>systemStatus</i>	// current status of the spacecraft																																																
<i>spacecraftOrbitData</i>	// vector of the orbital insertion data of all																																																
	// subswarm spacecraft																																																
<i>spacecraftTrajectoryData</i>	// vector of the trajectory data of all																																																
	// subswarm spacecraft																																																
<i>workerSpacecraft</i>	// vector of the <i>workers</i> in the subswarm																																																
<i>supplied fromSpacecraftID</i>	// spacecraft identification number that																																																
	// sent a message to be relayed																																																
<i>supplied toSpacecraftID</i>	// spacecraft identification number that																																																
	// is to received a relayed message																																																
<i>supplied messageRec</i>	// the message received to be relayed																																																
<i>messageRec</i>	// the message received to be relayed																																																
<i>messageHistory</i>	// history log of the messages sent																																																
<i>memoryStatus</i>	// current status (i.e., functioning or mal-functioning) of the spacecraft's memory																																																
	// subswarm spacecraft																																																
<i>spacecraftOrbitData</i>	// vector of the orbital insertion data of all																																																
	// subswarm spacecraft																																																
<i>spacecraftTrajectoryData</i>	// vector of the trajectory data of all																																																
	// subswarm spacecraft																																																
<i>messageHistory</i>	// history log of the messages sent																																																
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensure the timely delivery of message throughout the subswarm.																																																	
Safety - Ensure the accuracy of spacecraft orbital and trajectory insertion data to prevent spacecraft collisions.																																																	

Core Variation Point Schema for the Messenger Role

Role Schema: Messenger	Schema ID: M-LagrangeTravel														
Variation Point: Lagrange-Travel															
Inherits: M-Core															
Parameters of Variation: P21=True; P24=True															
Requirements: V_M2, V_M6															
Description: Provides the spacecraft with the ability to travel to the Lagrange point and communicate and with mission control in order to relay the data collected from scientific exploration.															
Activities and Protocols: CheckSystemStatus, GetArchiveData, PrepareCommToMisControl, TravelToLagrange, <u>AcceptMisConotrolConfirm</u> , <u>SendDataToMisControl</u>															
Permissions: Reads - <table style="margin-left: 40px; border: none;"> <tr> <td><i>systemStatus</i></td> <td>// current status of the spacecraft</td> </tr> <tr> <td><i>riskForSystemFactor</i></td> <td>// current risk to spacecraft to see if recent solar storm</td> </tr> <tr> <td><i>lagrangePointLocation</i></td> <td>// position of the Lagrange point to travel to</td> </tr> <tr> <td><i>solarSailStatus</i></td> <td>// current status of the solar sail</td> </tr> </table> Changes - <table style="margin-left: 40px; border: none;"> <tr> <td><i>systemStatus</i></td> <td>// status of the spacecraft</td> </tr> <tr> <td><i>solarSailStatus</i></td> <td>// status of the solar sail</td> </tr> </table> Generates - <table style="margin-left: 40px; border: none;"> <tr> <td><i>asteroidDataMsg</i></td> <td>// message to send to mission control containing the asteroid data collected</td> </tr> </table>		<i>systemStatus</i>	// current status of the spacecraft	<i>riskForSystemFactor</i>	// current risk to spacecraft to see if recent solar storm	<i>lagrangePointLocation</i>	// position of the Lagrange point to travel to	<i>solarSailStatus</i>	// current status of the solar sail	<i>systemStatus</i>	// status of the spacecraft	<i>solarSailStatus</i>	// status of the solar sail	<i>asteroidDataMsg</i>	// message to send to mission control containing the asteroid data collected
<i>systemStatus</i>	// current status of the spacecraft														
<i>riskForSystemFactor</i>	// current risk to spacecraft to see if recent solar storm														
<i>lagrangePointLocation</i>	// position of the Lagrange point to travel to														
<i>solarSailStatus</i>	// current status of the solar sail														
<i>systemStatus</i>	// status of the spacecraft														
<i>solarSailStatus</i>	// status of the solar sail														
<i>asteroidDataMsg</i>	// message to send to mission control containing the asteroid data collected														
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensure the successful delivery of data to mission control. Safety - None.															

Lagrange Traveler Variation Point Schema for the Messenger Role

Role Variation Points Schema: SelfHealer		Schemata ID: SH
Parameters of Variation: P2, P3		
Description:		
At the swarm-level, the collection of these roles within all the spacecraft aid in autonomously maintaining the system's scientific operations while enduring solar storms, spacecraft collisions, etc. At the spacecraft-level, detecting subsystem malfunctions and failures and autonomously reconfigure itself or requesting help to heal its damaged components.		
Variation Points:		
<u>Core:</u>	The spacecraft does not have the ability to alter its role (i.e., <i>worker</i> , <i>leader</i> or <i>messenger</i>) and only has the core functionality in regards to its self-healing ability. [SH-Core]	
UpToLeader:	The spacecraft has the ability to change from its current role, either <i>worker</i> or <i>messenger</i> , to that of a <i>leader</i> as a mechanism for swarm-level self-healing in the case that the swarm or subswarm needs to replace a lost or failing <i>leader</i> . [SSW-UpToLeader]	
UpToMessenger:	The spacecraft has the ability to change from its current role, either <i>worker</i> or <i>leader</i> , to that of a <i>messenger</i> as a mechanism for swarm-level self-healing in the case that the swarm or subswarm needs to replace a lost or failing <i>messenger</i> . [SSW-UpToMessenger]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time, however, the spacecraft may switch its operating variation point (e.g., the UpToLeader variation point) may be enacted at runtime. All spacecraft shall have the Basic variation point as a commonality.		

Self-Healer Role Variation Point Schema

Role Schema: SelfHealer	Schema ID: SH-Core
Variation Point: Core	
Inherits: None	
Parameters of Variation: P2=False; P3=False	
Requirements: C_G1, C_G2, C_SH1, C_SH2, C_SH3, C_SH4, C_SH5	
Description: Provides the spacecraft with the functionality to detect system damage and malfunctions and reconfigure itself so that it can continue pursuing its goals.	
Activities and Protocols: CheckMemoryStatus, CheckMemConfiguration, ReconfigMemory, RepairMemory, AcceptNewMemory, AcceptKillCommand, ReceiveNewMemoryReq, RequestNewMemory, SendNewMemory	
Permissions:	
Reads -	
<i>spacecraftID</i>	// spacecraft ID to send to other spacecraft
<i>memoryStatus</i>	// when requesting clean memory // current status (i.e., functioning or mal- // functioning) of the spacecraft's memory // system
<i>systemStatus</i>	// current status of the spacecraft
<i>riskForSystemFactor</i>	// current risk to spacecraft to see if recent // solar storm
Changes -	
<i>memoryCorrupted</i>	// Boolean value indicating if the spacecraft // believes its memory is corrupted
Generates -	
<i>memoryStatusReport</i>	// report containing information related to // the spacecraft's current memory status
<i>memoryCorruptedMsg</i>	// message indicating to other spacecraft // that its memory is damaged and needs // to reconfigure
Responsibilities:	
Liveness -	
If the spacecraft is functioning properly, this role will eventually be able to ensure that undamaged system memory is being used during scientific exploration.	
Safety -	
Preventing the spacecraft from using corrupted/damaged system memory and sending corrupted data to other spacecraft.	

Core Variation Point Schema for the Self-Healer Role

Role Schema: SelfHealer	Schema ID: SH-UpToLeader																		
Variation Point: UpToLeader																			
Inherits: SH-Core																			
Parameters of Variation: P2=True																			
Requirements: C_G1, C_G2, V_SH1																			
<p>Description: Provides the spacecraft with the functionalities needed to change itself from its current role, either a <i>messenger</i> or <i>worker</i>, to the role of a <i>leader</i>. This role change may be needed if too many <i>leader</i> spacecraft have been lost (e.g., due to collisions with other spacecraft or asteroids or due to solar storm damage) or are malfunctioning.</p>																			
<p>Activities and Protocols: CheckSystemStatus, EvaluateUpgradeReq, UpgradeToLeader, <u>AcceptUpgradeToLeader</u>, <u>ReceiveUpgradeRequest</u></p>																			
<p>Permissions:</p> <p>Reads -</p> <table> <tr> <td><i>systemStatus</i></td> <td>// current status of the spacecraft</td> </tr> <tr> <td><i>riskForSystemFactor</i></td> <td>// current risk to spacecraft</td> </tr> <tr> <td>supplied <i>upgradeGoal</i></td> <td>// goal provided to address the reason to upgrade to leader</td> </tr> <tr> <td>supplied <i>leadersVector</i></td> <td>// vector of other nearby leaders</td> </tr> </table> <p>Changes -</p> <table> <tr> <td><i>newSystemStatus</i></td> <td>// current status of the spacecraft</td> </tr> <tr> <td><i>riskForSystemFactor</i></td> <td>// current risk to spacecraft</td> </tr> <tr> <td><i>systemGoal</i></td> <td>// current goal of the spacecraft</td> </tr> </table> <p>Generates -</p> <table> <tr> <td><i>newSystemGoal</i></td> <td>// new goal of the spacecraft</td> </tr> <tr> <td><i>newSystemRole</i></td> <td>// new role for the spacecraft</td> </tr> </table>		<i>systemStatus</i>	// current status of the spacecraft	<i>riskForSystemFactor</i>	// current risk to spacecraft	supplied <i>upgradeGoal</i>	// goal provided to address the reason to upgrade to leader	supplied <i>leadersVector</i>	// vector of other nearby leaders	<i>newSystemStatus</i>	// current status of the spacecraft	<i>riskForSystemFactor</i>	// current risk to spacecraft	<i>systemGoal</i>	// current goal of the spacecraft	<i>newSystemGoal</i>	// new goal of the spacecraft	<i>newSystemRole</i>	// new role for the spacecraft
<i>systemStatus</i>	// current status of the spacecraft																		
<i>riskForSystemFactor</i>	// current risk to spacecraft																		
supplied <i>upgradeGoal</i>	// goal provided to address the reason to upgrade to leader																		
supplied <i>leadersVector</i>	// vector of other nearby leaders																		
<i>newSystemStatus</i>	// current status of the spacecraft																		
<i>riskForSystemFactor</i>	// current risk to spacecraft																		
<i>systemGoal</i>	// current goal of the spacecraft																		
<i>newSystemGoal</i>	// new goal of the spacecraft																		
<i>newSystemRole</i>	// new role for the spacecraft																		
<p>Responsibilities:</p> <p>Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensure that the current goal is satisfied or can be satisfied prior to upgrading to a new spacecraft role.</p> <p>Safety - Ensure that the subswarm has an adequate number of <i>leader</i> spacecraft.</p>																			

Promote to Leader Role Variation Point Schema for the Self-Healer Role

Role Schema: SelfHealer	Schema ID: SH-UpToMessenger																		
Variation Point: UpToMessenger																			
Inherits: SH-Core																			
Parameters of Variation: P3=True																			
Requirements: C_G1, C_G2, V_SH2, V_SH3																			
<p>Description: Provides the spacecraft with the functionalities needed to change itself from its current role, either a <i>messenger</i> or <i>worker</i>, to the role of a <i>leader</i>. This role change may be needed if too many <i>leader</i> spacecraft have been lost (e.g., due to collisions with other spacecraft or asteroids or due to solar storm damage) or are malfunctioning.</p>																			
<p>Activities and Protocols: CheckSystemStatus, EvaluateUpgradeReq, UpgradeToMessenger, AcceptUpgradeToMessenger, ReceiveUpgradeRequest</p>																			
<p>Permissions:</p> <p>Reads -</p> <table> <tr> <td><i>systemStatus</i></td> <td>// current status of the spacecraft</td> </tr> <tr> <td><i>riskForSystemFactor</i></td> <td>// current risk to spacecraft</td> </tr> <tr> <td>supplied <i>upgradeGoal</i></td> <td>// goal provided to address the reason to upgrade to leader</td> </tr> <tr> <td>supplied <i>messengersVector</i></td> <td>// vector of other nearby messengers</td> </tr> </table> <p>Changes -</p> <table> <tr> <td><i>newSystemStatus</i></td> <td>// current status of the spacecraft</td> </tr> <tr> <td><i>riskForSystemFactor</i></td> <td>// current risk to spacecraft</td> </tr> <tr> <td><i>systemGoal</i></td> <td>// current goal of the spacecraft</td> </tr> </table> <p>Generates -</p> <table> <tr> <td><i>newSystemGoal</i></td> <td>// new goal of the spacecraft</td> </tr> <tr> <td><i>newSystemRole</i></td> <td>// new role for the spacecraft</td> </tr> </table>		<i>systemStatus</i>	// current status of the spacecraft	<i>riskForSystemFactor</i>	// current risk to spacecraft	supplied <i>upgradeGoal</i>	// goal provided to address the reason to upgrade to leader	supplied <i>messengersVector</i>	// vector of other nearby messengers	<i>newSystemStatus</i>	// current status of the spacecraft	<i>riskForSystemFactor</i>	// current risk to spacecraft	<i>systemGoal</i>	// current goal of the spacecraft	<i>newSystemGoal</i>	// new goal of the spacecraft	<i>newSystemRole</i>	// new role for the spacecraft
<i>systemStatus</i>	// current status of the spacecraft																		
<i>riskForSystemFactor</i>	// current risk to spacecraft																		
supplied <i>upgradeGoal</i>	// goal provided to address the reason to upgrade to leader																		
supplied <i>messengersVector</i>	// vector of other nearby messengers																		
<i>newSystemStatus</i>	// current status of the spacecraft																		
<i>riskForSystemFactor</i>	// current risk to spacecraft																		
<i>systemGoal</i>	// current goal of the spacecraft																		
<i>newSystemGoal</i>	// new goal of the spacecraft																		
<i>newSystemRole</i>	// new role for the spacecraft																		
<p>Responsibilities:</p> <p>Liveness - If the spacecraft is functioning properly, this role will eventually be able to ensure that the current goal is satisfied or can be satisfied prior to upgrading to a new spacecraft role.</p> <p>Safety - Ensure that the subswarm has an adequate number of <i>messenger</i> spacecraft.</p>																			

Promote to Messenger Role Variation Point Schema for the Self-Healer Role

Role Variation Points Schema: SelfProtector		Schemata ID: SP
Parameters of Variation: N/A		
Description:		
At the swarm-level, the collection of these roles within all the spacecraft aid in autonomously maintaining the system's scientific operations while enduring solar storms, spacecraft collisions, etc. At the spacecraft-level, detecting subsystem malfunctions and failures and autonomously reconfigure itself or requesting help to heal its damaged components.		
Variation Points:		
SolarStormWarner:	Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm. [SSW]	
SolarStormProtector:	Protects the spacecraft from the solar radiation present during solar storms by using the solar sail as a shield, powering off systems and/or moving to a better position. [SSP]	
CollisionProtector:	Prevents the spacecraft from colliding with other spacecraft in the swarm and with nearby asteroids. [CP]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time,		

Self-Protector Role Variation Point Schema

Role Variation Points Schema: SolarStormWarner		Schemata ID: SSW
Parameters of Variation: P7, P8		
Description:		
Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm.		
Variation Points:		
<u>Passive:</u>	The spacecraft does not have the ability to constantly monitor the solar disc to watch for solar storms but can warn other spacecraft after itself receiving a warning message. [SSW-Passive]	
Warm-Spare:	The spacecraft has the ability to constantly monitor the solar disc to watch for solar storms and receive messages from mission control but is acting in a backup/redundant capacity. [SSW-Warm]	
Active:	The spacecraft is tasked to constantly monitor the solar disc and receive warning messages from mission control so that it can warn other spacecraft of an impending solar storm. [SSW-Active]	
Binding Time:		
The binding time to decide which variation point(s) a spacecraft has is at <i>design</i> time, however, the spacecraft may switch its operating variation point (e.g., from Warm-Spare to Active) at runtime. All spacecraft shall have the Passive variation point as a commonality. Spacecraft with the Warm-Spare variation point shall also include all functionality of Passive. Likewise, all spacecraft with the Active variation point shall have the functionality of the Warm-Spare.		

Self-Protector Solar Storm Warner Role Variation Point Schema

Role Schema: SolarStormWarner	Schema ID: SSW-Passive
Variation Point: Passive	
Inherits: SP-Core	
Parameters of Variation: P7=Passive; P8=False	
Requirements: C_G1, C_SH4, C_SP5, C_SP8, V_SP1, V_SP2	
Description: Receives warnings from other spacecraft about impending solar storms and calculates the risk factor to itself from solar radiation damage. Notifies other nearby spacecraft of the impending solar storm.	
Activities and Protocols: CalculateStormRisk, UpgradeToWarm, <u>AcceptUpgrade</u> , <u>AcceptWarnMsg</u> , <u>RecieveHeartbeat</u> , <u>ReplyHeartBeat</u> , <u>SendSolarStormWarnMsg</u>	
Permissions: Reads - <ul style="list-style-type: none"> <i>position</i> // current spacecraft position <i>velocityIncrement</i> // current spacecraft velocity increment <i>curScienceGoalFactor</i> // current spacecraft scientific goal factor <i>subswarmVector</i> // vector of nearby spacecraft to warn supplied <i>stormType</i> // type of storm supplied by warning supplied <i>stormIntensity</i> // storm intensity supplied by warning supplied <i>stormVector</i> // storm vector supplied by warning Changes - <ul style="list-style-type: none"> <i>riskForSystemFactor</i> // current risk to spacecraft Generates - <ul style="list-style-type: none"> <i>stormRiskValue</i> // new value of the risk to the spacecraft of // the solar storm 	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to optimize the ability to satisfy scientific goals while minimizing the risk factor. Safety - Prevent other spacecraft from being damaged by notifying others.	

Passive Variation Point Schema for the Solar Storm Warner Role

Role Schema: SolarStormWarner	Schema ID: SSW-Active
Variation Point: Active	
Inherits: SSW-Passive, SSW-Warm	
Parameters of Variation: P7=Active; P8=True	
Requirements: C_M9, V_SP1, V_SP2	
Description: Continuously monitors the solar disc for the signs of an impending solar storm whose solar radiation may damage the swarm's spacecraft. Upon detecting a solar storm, it seeks to verify the data and then proceeds to warn the swarm's spacecraft. Also able to receive warning messages from mission control of an impending solar storm.	
Activities and Protocols: CompareMissionControlData, DowngradeToWarm, <u>AcceptDowngrade</u> , <u>AcceptMissionControlWarn</u> , <u>AcceptStormDataVote</u> , <u>InitiateStormDataVote</u> , <u>InitiateStromWarning</u>	
Permissions: Reads - <i>detectedStormType</i> // type of storm as detected <i>detectedStormIntensity</i> // intensity of the storm as detected <i>detectedStormVector</i> // storm vector as detected supplied <i>MCStormType</i> // type of storm supplied by mission control supplied <i>MCStormIntensity</i> // storm intensity supplied by mission control supplied <i>MCstormVector</i> // storm vector supplied by mission control Changes - <i>stormRiskValue</i> // new value of the risk to the spacecraft of // the solar storm Generates - <i>riskForSystemFactor</i> // current risk to spacecraft <i>stromWarningConfidence</i> // confidence in the warning provided by // mission control <i>voteConfidence</i> // confidence in the verification of detected // storm data by other spacecraft <i>warningMessage</i> // warning message to be sent to other // spacecraft	
Responsibilities: Liveness - If the spacecraft is functioning properly, this role will eventually be able to maintain communication connection with mission control. Safety - Initiate warnings to spacecraft of an impending solar storm.	

Active Variation Point Schema for the Solar Storm Warner Role

Role Schema: SolarStormProtector	Schema ID: SSP
Variation Point: SolarStormProtector	
Inherits: None	
Parameters of Variation: N/A	
Requirements: C_SP5, C_SP6, C_SP7	
Description: Provides the spacecraft with the functionality to autonomously protect itself from the affects of solar radiation during a solar storm.	
Activities and Protocols: CheckSolarSailStatus, DeploySolarSailAsShield, EvaluateRiskToGoal, PowerDownSubsystems, PowerUpSubsystems	
Permissions:	
Reads -	
<i>curScienceGoalFactor</i>	// current spacecraft scientific goal factor
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
<i>solarSailStatus</i>	// status of the solar sail
<i>detectedStormType</i>	// type of storm as detected
<i>detectedStormIntensity</i>	// intensity of the storm as detected
<i>detectedStormVector</i>	// storm vector as detected
<i>subsystemsList</i>	// vector list of the spacecraft's subsystems
Changes -	
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
<i>systemStatus</i>	// status of the spacecraft
<i>solarSailStatus</i>	// status of the solar sail
<i>subsystemsStatus</i>	// list of the statuses of the spacecraft's subsystems
Generates -	
<i>riskForSystemFactor</i>	// current risk to spacecraft
<i>riskToGoalFactor</i>	// calculated value of the current risk factor to the advantage of pursuing scientific exploration
Responsibilities:	
Liveness - If the spacecraft is functioning properly, this role will eventually take the steps needed to prevent radiation damage from a solar storm.	
Safety - Prevent the solar radiation damage to the spacecraft possible during a solar storm.	

Protect from Solar Storms Role Variation Point Schema for the Self-Protector Role

Role Schema: CollisionProtector	Schema ID: CP
Variation Point: CollisionProtector	
Inherits: None	
Parameters of Variation: N/A	
Requirements: C_SP1, C_SP2, C_SP3, C_SP4, C_SP5	
Description: Provides the spacecraft with the functionality to autonomously protect itself from colliding with other spacecraft and nearby asteroids.	
Activities and Protocols: Analyze3DModel, DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids, MonitorNearbySpacecraft, MoveToAvoidCollision, AcceptAsteroid3DModel, AcceptCurrentPosition, AcceptCurrentTrajectory, AcceptSpacecraftLocations, NegotiateCollisionAvoidance, PingNearbySpacecraft, RequestAsteroidPositions, RequestCurrentPosition, RequestCurrentTrajectory, RequestSpacecraftLocations	
Permissions:	
Reads -	
<i>curScienceGoalFactor</i>	// current spacecraft scientific goal factor
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
supplied <i>asteroid3DModel</i>	// 3D model of an asteroid supplied
supplied <i>asteroidPositions</i>	// positions of nearby asteroids
supplied <i>subswarmVector</i>	// vector of nearby spacecraft positions
supplied <i>spacecraftPos</i>	// current position of a nearby spacecraft
	// supplied by a <i>messenger</i> or <i>leader</i>
supplied <i>spacecraftTraj</i>	// current trajectory of a nearby spacecraft
	// supplied by a <i>messenger</i> or <i>leader</i>
Changes -	
<i>position</i>	// current spacecraft position
<i>velocityIncrement</i>	// current spacecraft velocity increment
<i>riskForSystemFactor</i>	// current risk to spacecraft
Generates -	
<i>collisionRiskFactor</i>	// derived risk to spacecraft for an // impending collision
<i>riskToGoalFactor</i>	// calculated value of the current risk factor // to the advantage of pursuing scientific // exploration
<i>nearbyAsteroid</i>	// vector of nearby asteroids that must be // avoided to prevent a collision
<i>nearbySpacecraft</i>	// vector of nearby spacecraft that must be // avoided to prevent a collision
Responsibilities:	
Liveness -	
None.	
Safety -	
Prevent the collision with other spacecraft and nearby asteroids.	

Protect from Collisions Role Variation Point Schema for the Self-Protector Role

APPENDIX E. SOFTWARE FAILURE MODES, EFFECTS AND CRITICALITY ANALYSIS

This appendix provides the full set of Software Failure Modes, Effects and Criticality Analysis (SFMECA) tables for the Prospecting Asteroid Mission (PAM) multi-agent system product line (MAS-PL) case study used throughout this dissertation. The SFMECA tables are derived using the Gaia-PL requirements specifications schemas.

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	Analyze3DModel	Halt/Abnormal Termination	The position and model of a nearby asteroid stored in the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data vector may be incomplete or partially incorrect. This may affect other events such as MonitorNearbyAsteroids and MoveToAvoidCollision.	The spacecraft's inaccurate mental model of the nearby asteroid could cause it to maneuver itself too close to the asteroid causing a collision.	Major
			Omission	The role fails to analyze the 3D model of a nearby asteroid potentially causing the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data to be incomplete or incorrect. This may affect other events such as MonitorNearbyAsteroids and MoveToAvoidCollision.	The failure to analyze the 3D model provided of a nearby asteroid(s) may cause the asteroid to incorrectly maneuver itself too close to an asteroid and cause a collision.	Critical
			Incorrect Logic/Event	The role incorrectly analyzes the 3D model of a nearby asteroid that may cause the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data to be incomplete or incorrect. This may affect other events such as MonitorNearbyAsteroids and MoveToAvoidCollision.	The spacecraft uses an inaccurate 3D model of a nearby asteroid that may cause it to maneuver itself into a nearby spacecraft or asteroid.	Critical
			Timing/Order	The role fails to analyze the 3D model of a nearby asteroid causing the <i>asteroidPositions</i> , <i>nearbyAsteroid</i> and <i>collisionRiskFactor</i> data to be outdated. The <i>riskForSystemFactor</i> data may be inaccurate since it was calculated based on outdated data. This may affect other events such as MonitorNearbyAsteroids, EvaluateRiskToGoal and MoveToAvoidCollision.	The spacecraft uses an outdated 3D model of a nearby asteroid(s) and may not be able to react in time to avoid a collision with an asteroid if the 3D model is not updated as expected.	Major

SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	DetectNearby Spacecraft	Halt/Abnormal Termination	The role fails to complete its analysis of detecting nearby spacecraft and may not be aware of all nearby spacecraft. Thus, the data stored in <i>riskForSystemFactor</i> , <i>subswarmVector</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and <i>neabySpacecraft</i> may be inaccurate, corrupted or outdated.	The spacecraft does not have a full knowledge of all nearby spacecraft and may unknowingly maneuver itself into another spacecraft causing a collision. The spacecraft's ability to negotiate collision avoidance with another spacecraft using the NegotiateCollision Avoidance protocol can not be trusted by other spacecraft since the spacecraft's mental model of nearby spacecraft is not accurate.	Major
			Omission	The role fails to detect its surrounding for nearby spacecraft and may not be aware of all nearby spacecraft. The data stored in <i>riskForSystemFactor</i> , <i>subswarmVector</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and may be inaccurate or outdated and the <i>neabySpacecraft</i> may be incorrect or outdated.	The spacecraft has no knowledge of the positions of other nearby spacecrafts possibly causing the spacecraft to maneuver too close to another spacecraft causing a collision. The lack of knowledge of the positions of nearby spacecrafts may also cause the spacecraft's ability to avoid collisions using the Negotiate CollisionAvoidance protocol is using incomplete or inaccurate data.	Critical
			Incorrect Logic/Event	The role possible wrongly detects or miscalculates the positions of nearby spacecraft. The data stored in <i>riskForSystemFactor</i> , <i>subswarmVector</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and may be inaccurate or outdated and the <i>neabySpacecraft</i> may be incorrect or outdated.	The spacecraft's belief of the positions of other nearby spacecraft is inaccurate and it may collide into nearby spacecraft if maneuvers itself. The lack of knowledge of the positions of nearby spacecrafts may additionally cause the spacecraft's ability to avoid collisions using the Negotiate CollisionAvoidance protocol is using incomplete or inaccurate data.	Critical
			Timing/Order	The detection of nearby spacecrafts is delayed so that the role may not possible have the accurate locations of nearby spacecraft when it is expecting it. Because of this, the data stored in <i>riskForSystemFactor</i> , <i>spacecraftPos</i> , <i>collisionRiskFactor</i> and may be inaccurate or outdated and the <i>neabySpacecraft</i> may be incorrect or outdated without the role knowing this.	The spacecraft may believe that the positions of nearby spacecraft it has stored in <i>subswarmVector</i> and <i>spacecraftPos</i> is correct and thus may inadvertently maneuver too close to another spacecraft and collide into it. The spacecraft may also provide inaccurate information to other spacecraft using the NegotiateCollision Avoidance protocol that may result in further collisions of spacecraft.	Major

SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	MoveToAvoid Collision	Halt/Abnormal Termination	The <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data may be temporarily incorrect since the spacecraft did not complete moving to its new position. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft will not have moved to the position expected by other nearby spacecraft in the subswarm potentially causing a collision.	Major
			Omission	The spacecraft fails to move to its new assigned position in the subswarm possibly causing the <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data to be temporarily incorrect. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft will not have moved but, rather, maintain its previous position potentially causing a collision. Other spacecraft in the subswarm may expect the spacecraft to have moved to a new position which may cause a collision due to the discrepancies between actual and perceived spacecraft positions.	Critical
			Incorrect Logic/Event	The spacecraft fails to move to the position it is expecting possibly causing its <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data to be different than expected. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft moves to a position different that what it expects. Further, other spacecraft nearby will have expected the spacecraft to be in a different location potentially causing a collision.	Critical
			Timing/Order	The spacecraft fails to move to the new position until some later, undetermined time potentially causing its <i>position</i> , <i>velocityIncrement</i> and <i>collisionRiskFactor</i> data to be different than expected. This could potentially affect other events such as DetectNearby Spacecraft EvaluateRiskToGoal, and other protocols including NegotiateCollisionAvoidance.	The spacecraft fails to move to the position it indicated to other spacecraft via the NegotiateCollisionAvoidance protocol at the time expected by the other spacecraft. This may cause a collision.	Major

SFMECA Event Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	nearbyAsteroids	Incorrect Value	The variation point belief of the positions of nearby asteroids may be incorrect. The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect and the Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the wrong data. The <u>RequestAsteroidPositions</u> protocol may provide inaccurate information upon request.	The spacecraft will use incorrect values of the locations of nearby asteroids and may unknowingly maneuver too close to an asteroid and collide with it. The spacecraft may also provide the incorrect information to other spacecraft through the <u>RequestAsteroidPositions</u> protocol causing other spacecraft to potentially collide into an asteroid. The incorrect data may also invalidate the scientific data collected on the asteroids.	Critical
			Absent Value	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or corrupted since no location values for nearby asteroids were available. The Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the unavailable data.	The spacecraft will have no information on the location of nearby asteroids and will need to request the locations via the <u>RequestAsteroidPositions</u> protocol. May cause a collision with an asteroid since the locations are unknown. May corrupt some of the scientific data collected on the asteroids or cause the execution of the variation point to freeze.	Major
			Wrong Timing	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or outdated since the location of nearby asteroid data is old. The Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may result in outdated output.	The spacecraft may have made maneuvering decisions based on outdated information of the location of nearby asteroids. This may cause a collision with an asteroid since the locations are outdated.	Major
			Duplicated Value	The Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events may be uneedingly executed twice since the data was updated twice.	The spacecraft will may have had to execute the Analyze3DModel, EvaluateRiskToGoal and MoveToAvoidCollision events twice possibly delaying the response to request from other spacecraft.	Minor

SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	nearbySpacecraft	Incorrect Value	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or corrupted since no location values for other nearby spacecraft are available. The DetectNearbySpacecraft, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the incorrect.	The spacecraft will use incorrect values of the locations of nearby spacecraft and may unknowingly maneuver too close to another spacecraft and collide with it. The spacecraft may also provide the incorrect information to other spacecraft through the <u>RequestSpacecraftLocations</u> protocol causing other spacecraft to potentially collide.	Major
			Absent Value	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be missing or corrupted since no location values for other nearby spacecraft are available. The DetectNearbySpacecraft, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the unavailable data.	The spacecraft will have no information on the location of nearby spacecraft and will need to request the locations via the <u>RequestSpacecraftLocations</u> protocol. May cause a collision with an spacecraft since the locations are unknown.	Critical
			Wrong Timing	The <i>riskForSystemFactor</i> and <i>collisionRiskFactor</i> data may be incorrect or outdated since the location of nearby asteroid data is old. The DetectNearbySpacecraft, EvaluateRiskToGoal and MoveToAvoidCollision events may make wrong decisions or incorrect analysis based on the outdated data.	The spacecraft may have made maneuvering decisions based on outdated information of the location of nearby spacecraft. This may cause a collision since the locations are outdated.	Critical
			Duplicated Value	The EvaluateRiskToGoal and MoveToAvoidCollision events may be uneedingly executed twice since the data was updated twice.	The spacecraft may report to others that it is malfunctioning since it received duplicated values.	Minor

SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
Self-Protector						
	CollisionProtector	position	Incorrect Value	The variation point uses the incorrect value of its current position possibly affecting the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearby Asteroids and MoveToAvoidCollision events. This may also cause the variation point to incorrectly change its <i>riskForSystemFactor</i> data and generate inaccurate <i>collisionRiskFactor</i> , <i>riskToGoal Factor</i> , <i>nearby Asteroids</i> and <i>nearbySpacecraft</i> data.	The spacecraft does not know its actual position and may report a false position to other spacecraft via the <u>RequestSpacecraftLocations</u> protocol potentially causing a collision.	Major
			Absent Value	The missing or corrupted value of its current position may affect the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids and MoveToAvoidCollision events since the data is unusable. This may also cause the variation point to corrupted its <i>riskForSystemFactor</i> data and generate corrupted <i>collisionRiskFactor</i> , <i>riskToGoalFactor</i> , <i>nearbyAsteroids</i> and <i>nearbySpacecraft</i> data.	The spacecraft does not know its actual position and may report a false position to other spacecraft via the <u>RequestSpacecraftLocations</u> protocol potentially causing a collision. Alternatively, the spacecraft uses the missing or corrupted value and may collide into a nearby spacecraft.	Major
			Wrong Timing	The variation point uses the outdated value of its current position possibly affecting the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids and MoveToAvoidCollision events. This may also cause the variation point to incorrectly change its <i>riskForSystem Factor</i> data and generate outdated <i>collisionRiskFactor</i> , <i>riskToGoal Factor</i> , <i>nearbyAsteroids</i> and <i>nearbySpacecraft</i> data.	The spacecraft may have made maneuvering decisions based on outdated information of position potentially causing a collision.	Major
			Duplicated Value	The variation point uses the duplicate position information to execute the DetectNearbySpacecraft, EvaluateRiskToGoal, MonitorNearbyAsteroids and MoveToAvoidCollision events twice.	The spacecraft may report to others that it is malfunctioning since it received duplicated values of its current position.	Minor

SFMECA Data Table for the CollisionProtector Variation Point of the Self-Protector Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Passive	CalculateStorm Risk	Halt/Abnormal Termination	The <i>stormRiskValue</i> and the <i>riskForSystemFactor</i> data values may be incorrect or outdated since the role did not finish calculating the risk to the system of the impending solar storm.	The spacecraft may not take self-protection measure to guard against the solar radition from the impending solar storm. Alternatively, the spacecraft may take self-protection actions to guard again an impending solar storm when the actual risk to the spacecraft did not warrant such actions. This may unnecessarily consume power and/or increase the risk of collision.	Major
			Omission	The <i>stormRiskValue</i> and the <i>riskForSystemFactor</i> data values will not have been updated to reflect the received information of an impending solar storm.	The spacecraft may not take self-protection measure to guard against the solar radition from the impending solar storm.	Major
			Incorrect Logic/Event	The <i>stormRiskValue</i> and the <i>riskForSystemFactor</i> data values may be incorrect or corrupted since method of calculating them using the newly received information of an impending solar storm is incorrect.	The spacecraft may not take self-protection measure to guard against the solar radition from the impending solar storm. Alternatively, the spacecraft may take self-protection actions to guard again an impending solar storm when the actual risk to the spacecraft did not warrant such actions. This may unnecessarily consume power and/or increase the risk of collision.	Major
			Timing/Order	The <i>stormRiskValue</i> and the <i>riskForSystemFactor</i> data values may be outdated since the method of calculating them may not have executed before the impending solar storm arrived.	The spacecraft may not take self-protection actions in time to guard against the solar radition from the impending solar storm. Alternatively, the spacecraft may take self-protection actions to guard again an impending solar storm earlier than needed. This may unnecessarily consume power and/or increase the risk of collision.	Major

SFMECA Event Table for the Passive Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Passive	position	Incorrect Value	The variation point uses the incorrect value of its current position possibly affecting the CalculateStormRisk events. This may also cause the variation point to incorrectly change its <i>riskForSystemFactor</i> and <i>stormRiskValue</i> data.	The spacecraft may believe that, given its incorrect position, it is not at risk to an impending solar storm and may not take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm.	Major
			Absent Value	The variation point does not have a value for its current position possibly affecting the CalculateStormRisk events. This may also cause the variation point to corrupt its <i>riskForSystemFactor</i> and <i>stormRiskValue</i> data values.	The spacecraft may not know its current risk to an impending solar storm and may not take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm.	Major
			Wrong Timing	The variation point uses an outdated value of its current position possibly affecting the CalculateStormRisk events. This may also cause the variation point to incorrectly change or use outdated values to calculate its <i>riskForSystemFactor</i> and <i>stormRiskValue</i> data.	The spacecraft will have used an outdated position to calculate its risk and may not take self-protection measure to guard against the solar radiation from a impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm. Alternatively, the spacecraft may take self-protection actions to guard against an impending solar storm when the actual risk to the spacecraft did not warrant such actions. This may unnecessarily consume power and/or increase the risk of collision.	Major
			Duplicated Value	The variation point may use the duplicated position information to execute the CalculateStormRisk activity twice.	The spacecraft may have unnecessarily consume power or resources to execute activities twice.	Minor

SFMECA Data Table for the Passive Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Passive	stormIntensity	Incorrect Value	The variation point uses the incorrect value of the <i>sotrmIntensity</i> data that it received from another spacecraft as a part of the <u>AcceptWarnMsg</u> protocol. This may affect the CalculateStormRisk activity and can, in turn, incorrectly change the <i>riskForSystemFactor</i> and the <i>stormRiskValue</i> data as well as the <u>SendSolarStormWarnMsg</u> protocol.	The spacecraft calculates its risk to the impending solar storm incorrectly and may not take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm. Additionally, the spacecraft may relay the incorrect data using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft.	Critical
			Absent Value	The variation point receives a missing value the <i>sotrmIntensity</i> data that it received from another spacecraft as a part of the <u>AcceptWarnMsg</u> protocol potentially affecting the <i>riskForSystemFactor</i> and <i>stormRiskValue</i> data as well as the <u>SendSolarStormWarnMsg</u> protocol.	The spacecraft calculates its risk to the impending solar storm incorrectly and may not take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm. Additionally, the spacecraft may relay the missing data using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft.	Critical
			Wrong Timing	The variation point receives the <i>sotrmIntensity</i> data from another spacecraft via the <u>AcceptWarnMsg</u> protocol late. This may affect the calculation of the the CalculateStormRisk activity and can, in turn, may not be able to update the <i>riskForSystemFactor</i> and the <i>stormRiskValue</i> data in a timely manner.	The spacecraft may not calculate its risk of the impending solar storm in time to take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm. Additionally, the spacecraft might not relay the missing data using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft.	Major
			Duplicated Value	The variation point receives the <i>sotrmIntensity</i> data from another spacecraft via the <u>AcceptWarnMsg</u> protocol multiple times possible causing the variation point to execute the CalculateStormRisk activity multiple times.	The spacecraft might relay the warning message using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft multiple times.	Minor

SFMECA Data Table for the Passive Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Passive	stormVector	Incorrect Value	The variation point uses the incorrect value of the <i>stormVector</i> data that it received from another spacecraft as a part of the <u>AcceptWarnMsg</u> protocol. This may affect the CalculateStormRisk activity and can, in turn, incorrectly change the <i>riskForSystemFactor</i> and the <i>stormRiskValue</i> data as well as the <u>SendSolarStormWarnMsg</u> protocol.	The spacecraft calculates its risk to the impending solar storm incorrectly and may not take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm. Additionally, the spacecraft may relay the incorrect data using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft.	Critical
			Absent Value	The variation point receives a missing value the <i>stormVector</i> data that it received from another spacecraft as a part of the <u>AcceptWarnMsg</u> protocol potentially affecting the <i>riskForSystemFactor</i> and <i>stormRiskValue</i> data as well as the <u>SendSolarStormWarnMsg</u> protocol.	The spacecraft calculates its risk to the impending solar storm incorrectly and may not take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm. Additionally, the spacecraft may relay the missing data using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft.	Critical
			Wrong Timing	The variation point receives the <i>stormVector</i> data from another spacecraft via the <u>AcceptWarnMsg</u> protocol late. This may affect the calculation of the the CalculateStormRisk activity and can, in turn, may not be able to update the <i>riskForSystemFactor</i> and the <i>stormRiskValue</i> data in a timely manner.	The spacecraft may not calculate its risk of the impending solar storm in time to take self-protection measure to guard against the solar radiation from the impending solar storm. This may cause the spacecraft's memory to be corrupted and the data and/or the entire spacecraft to be lost to the swarm. Additionally, the spacecraft might not relay the missing data using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft.	Major
			Duplicated Value	The variation point receives the <i>stormVector</i> data from another spacecraft via the <u>AcceptWarnMsg</u> protocol multiple times possible causing the variation point to execute the CalculateStormRisk activity multiple times.	The spacecraft might relay the warning message using <u>SendSolarStormWarnMsg</u> protocol to other spacecraft multiple times.	Minor

SFMECA Data Table for the Passive Variation Point of the SolarStormWarner Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Warm-Spare	CalculateStormDataAccuracy	Halt/Abnormal Termination	The variation point does not finish executing the CalculateStormDataAccuracy to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate. This may affect the <i>stormDataAccuracyValue</i> and the <i>stormRiskValue</i> data. This may also affect the information sent in the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Omission	The variation point does not execute the CalculateStormDataAccuracy activity to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate. This may affect the <i>stormDataAccuracyValue</i> and the <i>stormRiskValue</i> data by rendering them incorrect, outdated or corrupted. This may also affect the information sent in the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Incorrect Logic/Event	The variation point incorrectly executes the CalculateStormDataAccuracy activity to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate. This may affect the <i>stormDataAccuracyValue</i> and the <i>stormRiskValue</i> data by rendering them incorrect. This may also affect the information sent in the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft. Alternatively, this may cause the spacecraft to agree with the information when it shouldn't which may cause an inadvertent warning message to be sent to the swarm.	Critical
			Timing/Order	The variation point the CalculateStormDataAccuracy activity to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate not in a timely manner. This may affect the <i>stormDataAccuracyValue</i> and the <i>stormRiskValue</i> data by rendering them incorrect. This may also affect the information sent in the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft votes too late as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical

SFMECA Event Table for the Warm-Spare Variation Point of the SolarStormWarner Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Warm-Spare	DetectStormData	Halt/Abnormal Termination	The variation point halts its detection of the solar storm data. This may affect the <i>stormData AccuracyValue</i> , <i>stormRiskValue</i> , <i>prelimStormType</i> , <i>prelimStorm Intensity</i> and <i>prelimStorm Vector</i> data rendering it incomplete, outdated or inaccurate. This may also affect the information sent in the <u>Vote StormDataAccuracy</u> protocol.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Omission	The variation point fails to detect the solar storm data. This may affect the <i>stormDataAccuracy Value</i> , <i>stormRiskValue</i> , <i>prelim StormType</i> , <i>prelimStorm Intensity</i> and <i>prelimStorm Vector</i> data rendering it incomplete, outdated or inaccurate. This may also affect the information sent in the <u>Vote StormDataAccuracy</u> protocol.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Incorrect Logic/Event	The variation point uses incorrect logic to detect the solar storm data. This may affect the <i>stormData AccuracyValue</i> , <i>storm RiskValue</i> , <i>prelimStorm Type</i> , <i>prelimStorm Intensity</i> and <i>prelimStorm Vector</i> data rendering it in-accurate. This may also affect the information sent in the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft. Alternatively, this may cause the spacecraft to agree with the information when it shouldn't which may cause an inadvertent warning message to be sent to the swarm.	Critical
			Timing/Order	The variation point detects the solar storm data not in a timely manner. This may affect the <i>stormData AccuracyValue</i> , <i>stormRiskValue</i> , <i>prelimStorm Type</i> , <i>prelimStorm Intensity</i> and <i>prelimStorm Vector</i> data rendering it inaccurate. This may also affect the information sent in the <u>Vote StormDataAccuracy</u> protocol.	The spacecraft votes too late as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical

SFMECA Event Table for the Warm-Spare Variation Point of the SolarStormWarner Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Warm-Spare	ObserveSolarDisc	Halt/Abnormal Termination	The variation point halts its observation of the solar disc. This may affect the <i>stormData AccuracyValue</i> , <i>stormRiskValue</i> , <i>prelimStormType</i> , <i>prelimStorm Intensity</i> and <i>prelimStom Vector</i> data rendering it incomplete, outdated or inaccurate. This may also affect the information sent in the <u>Vote StormDataAccuracy</u> protocol.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Omission	The fails to observe the solar disc. This may affect the <i>stormDataAccuracy Value</i> , <i>stormRiskValue</i> , <i>prelim StormType</i> , <i>prelimStorm Intensity</i> and <i>prelimStom Vector</i> data rendering it incomplete, outdated or inaccurate. This may also affect the information sent in the <u>Vote StormDataAccuracy</u> protocol.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Incorrect Logic/Event	The variation point uses incorrect logic to observe the solar disc. This may affect the <i>stormData AccuracyValue</i> , <i>storm RiskValue</i> , <i>prelimStorm Type</i> , <i>prelimStorm Intensity</i> and <i>prelimStom Vector</i> data rendering it inaccurate. This may also affect the information sent in the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft votes with inaccurate information as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft. Alternatively, this may cause the spacecraft to agree with the information when it shouldn't which may cause an inadvertent warning message to be sent to the swarm.	Critical
			Timing/Order	The fails to observe the solar disc when it should. This may affect the <i>stormData AccuracyValue</i> , <i>stormRiskValue</i> , <i>prelimStorm Type</i> , <i>prelimStormIntensity</i> and <i>prelimStomVector</i> data rendering it inaccurate. This may also affect the information sent in the <u>VoteStormData Accuracy</u> protocol.	The spacecraft votes too late as to whether it agrees with the solar storm information detected by another spacecraft. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical

SFMECA Event Table for the Warm-Spare Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Warm-Spare	<i>prelimStormIntensity</i>	Incorrect Value	The variation point uses an incorrect value for the <i>prelimStormIntensity</i> data supplied by another spacecraft possibly affecting its calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft may incorrectly judge the information provided as not an impending solar storm and thus not agree with sending out a warning. This may cause no warnings to be sent to the spacecraft of the swarm causing spacecraft to be lost due to the impending solar storm's radiation.	Critical
			Absent Value	The <i>prelimStormIntensity</i> data supplied by another spacecraft is missing which could possibly affect the variation point's calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft may incorrectly judge the information provided as not an impending solar storm and thus not agree with sending out a warning. This may cause no warnings to be sent to the spacecraft of the swarm causing spacecraft to be lost due to the impending solar storm's radiation.	Critical
			Wrong Timing	The variation point uses the value for the <i>prelimStormIntensity</i> data supplied by another spacecraft at the wrong time possibly affecting its calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft may incorrectly judge the information provided as not an impending solar storm and thus not agree with sending out a warning. This may cause a delay or no warnings to be sent to the spacecraft of the swarm causing spacecraft to be lost due to the impending solar storm's radiation.	Critical
			Duplicated Value	The <i>prelimStormIntensity</i> data supplied by another spacecraft possibly is used twice possibly affecting its calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol by possibly sending redundant messages.	The spacecraft may redundantly reply to the message from the spacecraft giving its agreement or disagreement to the solar storm information.	Minor

SFMECA Data Table for the Warm-Spare Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Warm-Spare	<i>prelimStormVector</i>	Incorrect Value	The variation point uses an incorrect value for the <i>prelimStormVector</i> data supplied by another spacecraft possibly affecting its calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft may incorrectly judge the information provided as not an impending solar storm and thus not agree with sending out a warning. This may cause no warnings to be sent to the spacecraft of the swarm causing spacecraft to be lost due to the impending solar storm's radiation.	Critical
			Absent Value	The <i>prelimStormVector</i> data supplied by another spacecraft is missing which could possibly affect the variation point's calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft may incorrectly judge the information provided as not an impending solar storm and thus not agree with sending out a warning. This may cause no warnings to be sent to the spacecraft of the swarm causing spacecraft to be lost due to the impending solar storm's radiation.	Critical
			Wrong Timing	The variation point uses the value for the <i>prelimStormVector</i> data supplied by another spacecraft at the wrong time possibly affecting its calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol.	The spacecraft may incorrectly judge the information provided as not an impending solar storm and thus not agree with sending out a warning. This may cause a delay or no warnings to be sent to the spacecraft of the swarm causing spacecraft to be lost due to the impending solar storm's radiation.	Critical
			Duplicated Value	The <i>prelimStormVector</i> data supplied by another spacecraft possibly is used twice possibly affecting its calculations in the CalculateStormDataAccuracy, CompareVerifyStormData, DetectStormData, ObserveSolarDisc activities. Further, this may affect the information it sends using the <u>VoteStormDataAccuracy</u> protocol by possibly sending redundant messages.	The spacecraft may redundantly reply to the message from the spacecraft giving its agreement or disagreement to the solar storm information.	Minor

SFMECA Data Table for the Warm-Spare Variation Point of the SolarStormWarner Role

Role	Variation Point	Event	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Active	CompareMissionControlData	Halt/Abnormal Termination	The variation point does not finish executing the CompareMissionControlData to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , and the <i>stormRiskValue</i> data. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the information sent by mission control. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Omission	The variation point does not execute the CompareMissionControlData activity to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , and the <i>stormRiskValue</i> data. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft fails to vote or votes with inaccurate information as to whether it agrees with the information detected by mission control. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical
			Incorrect Logic/Event	The variation point incorrectly executes the CompareMissionControlData activity to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , and the <i>stormRiskValue</i> data. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft votes with inaccurate information as to whether it agrees with the information detected by mission control. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft. Alternatively, this may cause the spacecraft to agree with the information when it shouldn't which may cause an inadvertent warning message to be sent to the swarm.	Critical
			Timing/Order	The variation point the CompareMissionControlData activity to determine if its detected data and another spacecraft's detected data regarding an impending solar storm are accurate not in a timely manner. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , and the <i>stormRiskValue</i> data. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft votes too late as to whether it agrees with the information detected by mission control. This may delay or prevent a warning message to be generated and sent to the swarm warning of an impending solar storm which may result in the loss of spacecraft.	Critical

SFMECA Event Table for the Active Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Active	<i>detectedStorm Intensity</i>	Incorrect Value	The variation point uses an incorrect value for the <i>detectedStormIntensity</i> data detected by the spacecraft and used to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data as well as the CompareMissionControl Data activity. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may provide the incorrect data to other spacecraft monitoring the solar disc potentially leading to the failure to issue a warning to the spacecraft of the subwarm of an impending solar storm. This may cause the loss of several spacecraft as a result of solar radiation damage. Alternatively, the incorrect data could lead to issuing a storm warning when one is not needed.	Critical
			Absent Value	The variation point uses missing value for the <i>detectedStormIntensity</i> data detected by the spacecraft and used to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data by corrupting them or rendering them inaccurate as well as the CompareMissionControl Data activity. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may provide the missing data to other spacecraft monitoring the solar disc potentially leading to the failure to issue a warning to the spacecraft of the subwarm of an impending storm. This may cause the loss of several spacecraft as a result of radiation damage. Or, the missing data could lead to issuing a solar storm warning when it is not needed.	Critical
			Wrong Timing	The variation point uses the <i>detectedStorm Intensity</i> data detected by the spacecraft at the wrong time to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data as well as the CompareMissionControl Data activity. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may fail to issue a warning message to the swarm of an impending solar storm in time to allow the spacecraft to take appropriate self-protection actions. This may cause the loss of several spacecraft as a result of solar radiation damage.	Critical
			Duplicated Value	The variation point uses duplicated value for the <i>detectedStormIntensity</i> data detected by the spacecraft and used to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data as well as the CompareMissionControlData activity by executing or defining it multiple times. This may also affect the information sent in the <u>InitiateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may redundantly issue a message to other spacecraft monitoring the solar disc seeking confirmation of the data they detected.	Minor

SFMECA Data Table for the Active Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Active	<i>detectedStormVector</i>	Incorrect Value	The variation point uses an incorrect value for the <i>detectedStormVector</i> data detected by the spacecraft and used to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRiskValue</i> data as well as the Compare MissionControl Data activity. This may also affect the information sent in the <u>Iniate StormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may provide the incorrect data to other spacecraft monitoring the solar disc potentially leading to the failure to issue a warning to the spacecraft of the subwarm of an impending solar storm. This may cause the loss of several spacecraft as a result of solar radiation damage. Or, the incorrect data could lead to issuing a storm warning when one is not needed.	Critical
			Absent Value	The variation point uses missing value for the <i>detectedStormVector</i> data detected by the spacecraft and used to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRiskValue</i> data by corrupting them or rendering them inaccurate as well as the CompareMissionControl Data activity. This may also affect the information sent in the <u>IniateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may provide the missing data to other spacecraft monitoring the solar disc potentially leading to the failure to issue a warning to the spacecraft of the subwarm of an impending storm. This may cause the loss of several spacecraft as a result of radiation damage. Or, the missing data could lead to issuing a solar storm warning when it is not needed.	Critical
			Wrong Timing	The variation point uses the <i>detectedStormVector</i> data detected by the spacecraft at the wrong time to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRiskValue</i> data as well as the Compare MissionControl Data activity. This may also affect the information sent in the <u>IniateStormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may fail to issue a warning message to the swarm of an impending solar storm in time to allow the spacecraft to take appropriate self-protection actions. This may cause the loss of several spacecraft as a result of solar radiation damage.	Critical
			Duplicated Value	The variation point uses duplicated value for the <i>detectedStormVector</i> data detected by the spacecraft and used to send to other spacecraft for verification. This may affect the <i>stormWarningConfidence</i> , <i>voteConfidence</i> , <i>warningMessage</i> , and the <i>stormRiskValue</i> data as well as the CompareMissionControl Data activity by executing or defining it multiple times. This may also affect the information sent in the <u>Iniate StormDataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft may redundantly issue a message to other spacecraft monitoring the solar disc seeking confirmation of the data they detected.	Minor

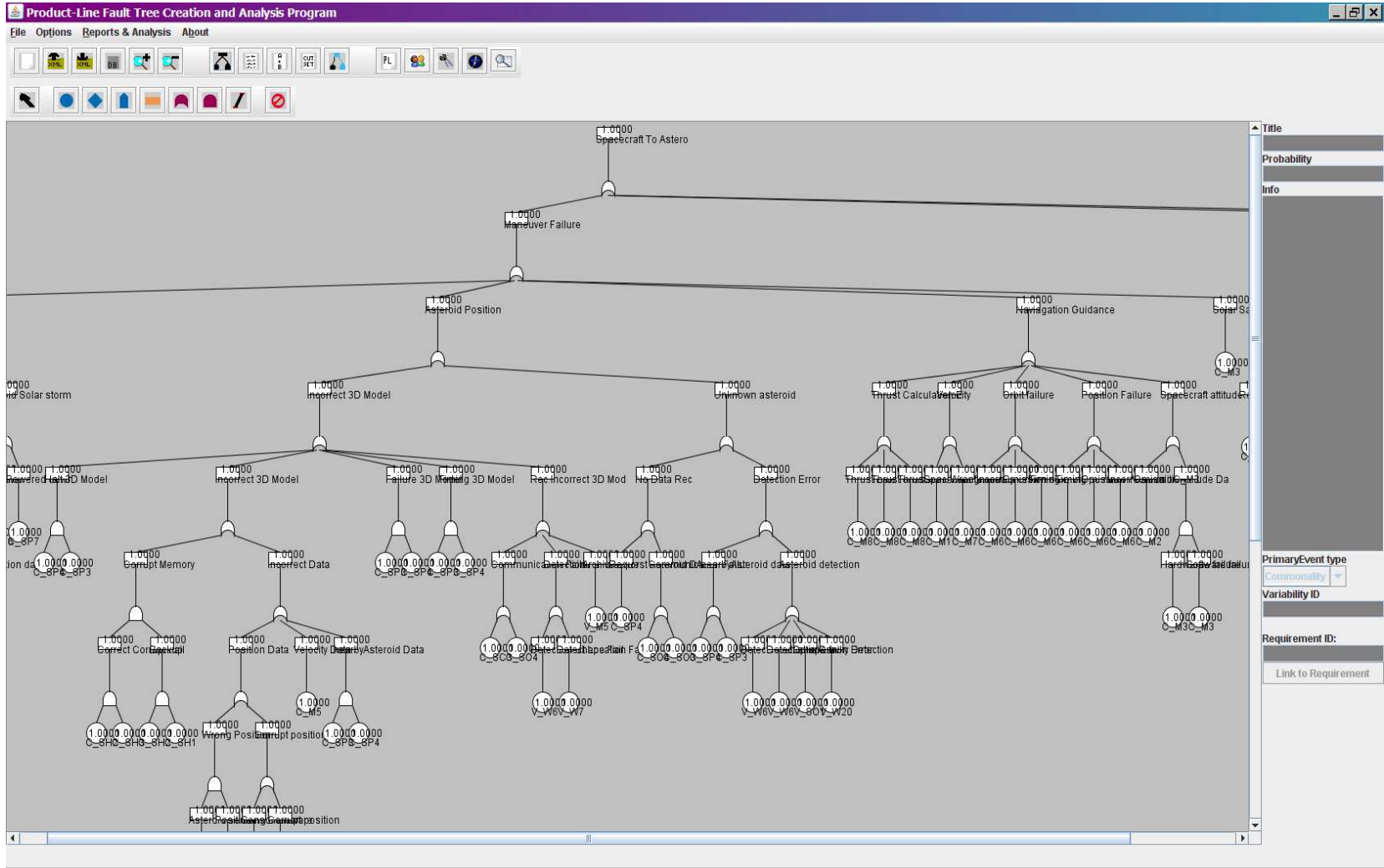
SFMECA Data Table for the Active Variation Point of the SolarStormWarner Role

Role	Variation Point	Data	Failure Mode	Local Effect(s)	System Effect(s)	Criticality
SolarStormWarner						
	Active	<i>warning Message</i>	Incorrect Value	The variation point uses an incorrect value for the <i>warningMessage</i> data generated by the spacecraft and used to send to other spacecraft as a warning of an impending solar storm. This may affect the <i>storm Warning Confidence</i> , <i>vote Confidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data. This may also affect the information sent in the <u>Iniate Storm DataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft monitoring the solar disc issues a warning message to the spacecraft of the swarm containing incorrect information. This may cause spacecraft to not take self-protection actions to protect from an impending solar storm when it should. This may cause the loss of several spacecraft as a result of radiation damage. Alternatively, it may cause spapcecraft to take self-protection actions when not needed.	Critical
			Absent Value	The variation point uses an empty value for the <i>warningMessage</i> data generated by the spacecraft and used to send to other spacecraft as a warning of an impending solar storm. This may affect the <i>storm Warning Confidence</i> , <i>vote Confidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data. This may also affect the information sent in the <u>Iniate Storm DataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft monitoring the solar disc issues a warning message to the spacecraft of the swarm containing missing information. This may cause spacecraft to not take self-protection actions to protect from an impending solar storm when it should. This may cause the loss of several spacecraft as a result of radiation damage. Alternatively, it may cause spapcecraft to take self-protection actions when not needed.	Critical
			Wrong Timing	The variation point issues the <i>warningMessage</i> generated by the spacecraft and used to send to other spacecraft as a warning of an impending solar storm not at the appropriate time. This may affect the <i>storm Warning Confidence</i> , <i>vote Confidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data. This may also affect the information sent in the <u>Iniate Storm DataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft monitoring the solar disc issues a warning message to the spacecraft of the swarm not in a timely manner. This may cause spacecraft to not take self-protection actions to protect from an impending solar storm when it should. This may cause the loss of several spacecraft as a result of radiation damage.	Critical
			Duplicated Value	The variation point generates multiple, redundand <i>warningMessage</i> message to send to other spacecraft as a warning of an impending solar storm. This may affect the <i>storm Warning Confidence</i> , <i>vote Confidence</i> , <i>warningMessage</i> , and the <i>stormRisk Value</i> data. This may also affect the information sent in the <u>Iniate Storm DataVote</u> and <u>InitiateStormWarning</u> protocols.	The spacecraft monitoring the solar disc issues multiple warning message to the spacecraft of the swarm.	Minor

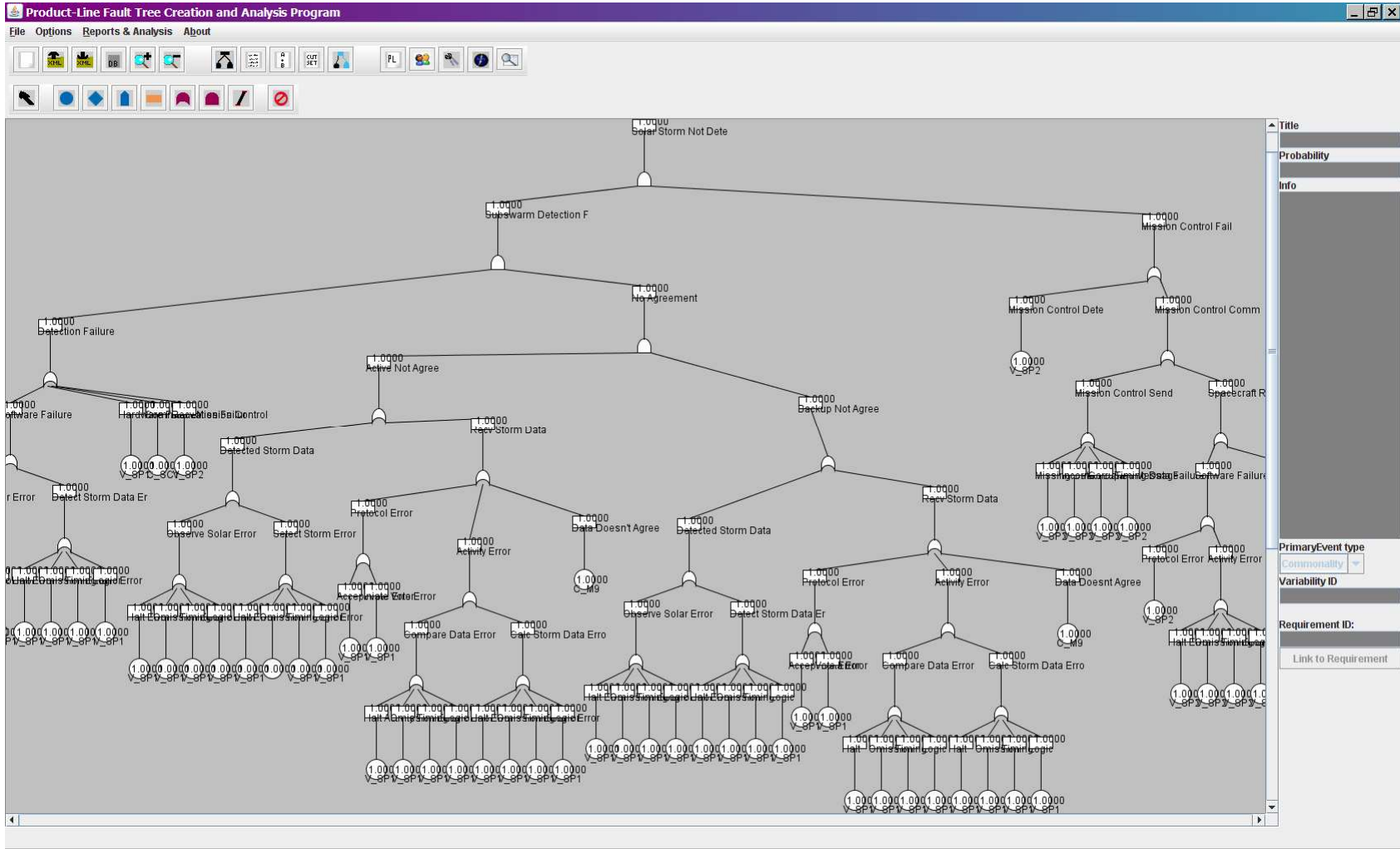
SFMECA Data Table for the Active Variation Point of the SolarStormWarner Role

APPENDIX F. PRODUCT-LINE SOFTWARE FAULT TREE ANALYSIS

This appendix provides screenshots of the Product-Line Software Fault Tree Analysis (PL-SFTA) constructed in PLFaultCAT for the Prospecting Asteroid Mission (PAM) multi-agent system product line (MAS-PL) case study used throughout this dissertation. The leaf nodes of the PL-SFTA fault trees associate to the product-line commonality and/or variability requirements described in the Commonality and Variability Analysis (CVA).



PL-SFTA for the Spacecraft to Asteroid Collision Hazard



PL-SFTA for the Failure to Detect Impending Solar Storm Hazard