
By

Roselane Santana Silva

B.Sc. Dissertation

Federal University of Bahia
cceagmat@ufba.br
wiki.dcc.ufba.br/DCC/

Salvador, BA- October, 2016
Roselane Santana Silva


A B.Sc. Dissertation presented to the Computer Science Department of Federal University of Bahia in partial fulfillment of the requirements for the degree of Bachelor in Computer Science.

Advisor: Eduardo Santana de Almeida
Co-Advisor: John D. McGregor

Salvador, BA- October, 2016
I dedicate this dissertation to God, my family, friends and professors who gave me all necessary support to get here.
First of all, I would like to thank the One who is worthy of all honor, Jesus.

I am eternally grateful for my beautiful family and friends. Especially my parents, my dad who is my hero, who has always supported my dreams, and takes care of me; as well as my super mom who sacrificed her own happiness just to see mine, and showed me how strong of a woman she is when I moved away for my studies abroad. Also, my two older brothers, Robson and Edson, who always protected me from everything and advised me on every single decision I would make. I love you all so much.

I could not forget to express my appreciation to Dr. Eduardo Almeida. He was the first professor, at UFBA, to believe in my capacity, never gave up on me, and kept me on track during my entire academic journey. He is someone who I truly admire and has always been an inspirational figure to me.

I am also in eternal debt to Dr. John McGregor for accepting me as a summer intern to work under his wing at Clemson University, and staying in touch ever since. Besides being an excellent professional who patiently guided me along the way as I wrote my bachelor’s thesis, John became a great friend of mine.

Finally, I would not have gotten here if it was not for my professors, my UFBA friends, my second family from the Science Without Borders program, my coworkers at PRODEB, and my best friends who I do not dare to cite names as they are all very important to me.

To all of you, thank you so much. <3
For from Him and through Him and to Him are all things.
To Him be the glory forever.
— ROMANS 11:36
Sistemas críticos estão cada vez mais integrados a nossa vida cotidiana. Nós dependemos de softwares para automação, como o monitoramento médico, e nós dependemos cada vez mais de softwares para tomar decisões de alto risco, cenários críticos para a segurança, onde pode não haver tempo suficiente para alertar os seres humanos, tais como veículos com piloto automático.

Estes sistemas dependem de técnicas complexas e experiência de domínio para avaliar a segurança. Os métodos tradicionais para a criação de sistemas críticos são caros, demorados e não adequados para produtos de consumo, tais como as "coisas" conectadas à Internet das Coisas (IoT).

Engenharia de Software de sistemas críticos requer uma compreensão clara do sistema. Engenharia de Linha de Produtos tem sido bem sucedida em reduzir o tempo e o custo de criação e implantação de produtos tradicionais através da reutilização sistemática de artefatos da linha de produto. No entanto, é um desafio único gerenciar questões de segurança em linhas de produtos de sistemas críticos, devido às muitas possibilidades de interações entre funcionalidades, fluxos de dados, e por assumir que o sistema não pode falhar.

Para apoiar essas questões, este trabalho apresenta uma primeira definição de uma abordagem para a arquitetura, design, implementação e realização de análise de segurança em linhas de produtos de software de sistemas críticos (SCSPL). A abordagem proposta é baseada na Linguagem de Design e Análise de Arquitetura (AADL), que é adequada para descrever arquitetura de sistemas críticos.

Nós ilustramos a abordagem proposta com uma linha de produto de sistemas de piloto automático, e reportamos os resultados deste projeto piloto, que se destina a estabelecer as bases para um programa de estudo na análise de segurança em Linhas de Produtos de Software de Sistemas Críticos (SCSPL).

**Palavras-chave:** Linhas de Produto de Software, Sistemas Críticos, Linguagem de Design e Análise de Arquitetura
Safety-critical software is increasingly integrated into everyday life. We depend on software for automation, such as medical monitoring, and we increasingly rely on software to make decisions in high-risk, safety-critical scenarios where there may not be enough time to alert humans, such as autonomous vehicle navigation. These systems depend on complex techniques and domain expertise to support safety assessment. Traditional methods for creating safety-critical software are expensive, time-consuming and not suitable for consumer products such as the "things" attached to the Internet of Things (IoT). Software engineering of safety-critical systems requires a clear understanding of the system. Product-line engineering has proven successful at reducing the cost and the time of creating and deploying traditional products through systematic reuse of product-line assets. However, it is a unique challenge to manage safety issues in safety-critical of product lines due to its many possibilities of features interactions, data flows, and assuming that the system can not fail.

To support it, this paper presents an initial definition of an approach for architecting, analyzing, implementing, and conducting safety analysis in safety-critical software product lines (SCSPL). The proposed approach is based on the Architecture Analysis and Design Language (AADL), which is suitable to describe the system’s architecture.

We illustrate their use on a family of Cruise Control Systems, safety-critical systems in the automotive domain, and report the results of this pilot project, which is intended to lay the foundation for a program of study on safety analysis for SCSPL.

Keywords: Software Product Lines, Safety-Critical Systems, Safety Analysis Techniques, Architecture Analysis & Design Language.
# Contents

List of Figures xi

List of Tables xii

List of Acronyms xiii

1 Introduction 1

1.1 Motivation ................................................. 2
1.2 Problem Statement ........................................... 2
1.3 Statement of the Contributions ................................. 2
1.4 Research Design ............................................. 3
1.5 Related Work ................................................ 4
1.6 Dissertation Structure ......................................... 5

2 Background 7

2.1 Introduction .................................................. 7
2.2 Safety-Critical Systems (SCS) ................................. 7
2.3 Software Product Lines (SPL) ................................. 8
2.4 Safety-Critical Software Product Lines (SCSPL) ............... 8
2.5 Existing Hazard Analysis Techniques ......................... 9
2.5.1 Software Fault Tree Analysis (SFTA) ....................... 9
2.5.2 Software Failure Mode, Effects and Analysis (SFMEA) .................. 10
2.5.3 System-Theoretic Process Analysis (STPA) .................. 10
2.6 Architecture Analysis and Design Language (AADL) .......... 10
2.7 Chapter Summary .............................................. 11

3 The Product Line Architecture Model 12

3.1 Introduction .................................................. 12
3.2 Overview of the example problem ............................... 12
3.2.1 Cruise Control (CC) ....................................... 12
3.2.2 Adaptive Cruise Control (ACC) ............................ 13
3.2.3 Collaborating Adaptive Cruise Control (CACC) ............... 13
3.3 Architecture Analysis and Design Language (AADL) .......... 14
3.4 Assume Guarantee REasoning Environment (AGREE) ........... 14
3.5 Error Modeling Annex ......................................... 16
### 3.6 Chapter Summary

---

### 4 Conducting a Hazard Analysis Using STPA

#### 4.1 Introduction

---

#### 4.2 Hazard Definition

---

#### 4.3 Description of STPA

- **4.3.1 STPA Report Concepts**
  - Background
  - STPA Fundamentals

- **Unsafe Control Actions**

---

#### 4.4 Analyzing the Cruise Control SPL using STPA

- **4.4.1 Background**

- **Assumptions**

- **4.4.2 Preliminaries**
  - System Boundaries
  - Accident Levels
  - Accidents
  - Hazards
  - Safety Constraints
  - Control Actions
  - Process Model

- **4.4.3 Unsafe Control Actions**
  - Causes and Compensations

---

#### 4.5 Chapter Summary

---

### 5 Safety Critical Design Patterns

#### 5.1 Introduction

---

#### 5.2 Motivation

---

#### 5.3 Redundancy in hardware and software

---

#### 5.4 Applying the patterns

- **5.4.1 Pattern Description**
- **5.4.2 Protected Single Channel Pattern (PSCP)**
- **5.4.3 Heterogeneous Redundancy Pattern (HRP)**

---

#### 5.5 Product Line Architecture

---

#### 5.6 Chapter Summary

---
6 An approach to implementing safety analysis in SCSPL

6.1 Introduction ................................................. 33

6.2 Issues in composing safety critical components into a SCSPL . . . . . . . . . . 33

6.3 A Description of the Safety Analysis Technique .................. 34

   6.3.1 Basic component error models ......................... 34
       Identify basic sources of error behavior .............. 34
       Cover the Error Ontology ............................... 34
       Identify Sinks and Propagations ....................... 36
       Select Propagating Error Flows ....................... 36

   6.3.2 Composite Error Models ............................... 37
       Integrate Component Error Models ..................... 37
       Specialize Errors ....................................... 38
       Identify Sinks and Propagations ....................... 38
       Integrate Error Models .................................. 38

   6.3.3 Design top level error behavior ...................... 38

6.4 Discussion of the technique ................................. 40

   6.4.1 Benefits of Using the Error Model Annex .............. 40
   6.4.2 Limitations of Using the Error Model Annex .......... 41

6.5 Chapter Summary ............................................. 41

7 Conclusion ....................................................... 42

   7.1 Lessons Learned .......................................... 42
   7.2 Future Work .............................................. 43

References ......................................................... 44

Appendix ............................................................. 48

A Cruise Control Safety Requirements .............................. 50

   A.1 Cruise Control (CC) ...................................... 50
   A.2 Requirements Definition and Analysis Language (RADL) .... 50
   A.3 State Machine .......................................... 51
List of Figures

3.1 A graphical view of an AADL component ......................... 15
3.2 Excerpt from an AGREE contract of the User Interface Component . . 16
3.3 Timing related errors in the EMV2 error type hierarchy .. 17

4.1 Hazard control process ........................................... 19
4.2 The New York Times | Florida Traffic Crash Report ............... 21
4.3 The Process Model for ACC Module ................................. 24

5.1 Single Design Pattern Example ...................................... 28
5.2 Dual Channel Pattern Level 1 ....................................... 31
5.3 Dual Channel Pattern Level 2 ....................................... 31

6.1 Component Error Behavior ........................................... 35
6.2 Error Ontology Hierarchy ........................................... 35
6.3 Error propagation for Generic Sensor ............................... 36
6.4 Compositional Error Behavior for ACC ............................. 37
6.5 Radar end-to-end Flow .............................................. 38
6.6 Error propagations leading to hazard ............................. 39
6.7 Two components being integrated .................................. 39

A.1 Requirements Definition and Analysis Language (RADL) Model ..... 51
A.2 CC State Machine ................................................ 51
A.3 ACC State Machine ................................................ 52
A.4 CACC State Machine ............................................... 52
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Product Map</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>Excerpt from Radar Unsafe Control Actions</td>
<td>25</td>
</tr>
</tbody>
</table>
List of Acronyms

AAA American Automobile Association
ACC Adaptive Cruise Control
AADL Architecture Analysis and Design Language
AGREE Assume Guarantee REasoning Environment
CACC Collaborating Adaptive Cruise Control
CC Cruise Control
CPS Cyber-Physical Systems
CRC Class Responsibility Collaboration
DoT Department of Transportation
EMV2 Error Modeling Annex (version 2)
FAA Federal Aviation Administration
HRP Heterogeneous Redundancy Pattern
IoT Internet of Things
NASA National Aeronautics and Space Administration
NHTSA National Highway Traffic Safety Administration
OSATE Open Source AADL Tool Environment
PSCP The Protected Single Channel Pattern
RADL Requirements Definition and Analysis Language
SAE Society of Automotive Engineers
SCS Safety-Critical Systems
SCSPL Safety-Critical Software Product Lines
SE Software Engineering
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL</td>
<td>Software Product Lines</td>
</tr>
<tr>
<td>SFTA</td>
<td>Software Fault Tree Analysis</td>
</tr>
<tr>
<td>SFMEA</td>
<td>Software Failure Mode, Effects and Analysis</td>
</tr>
<tr>
<td>STPA</td>
<td>System-Theoretic Process Analysis</td>
</tr>
<tr>
<td>V2V</td>
<td>vehicle-to-vehicle</td>
</tr>
</tbody>
</table>
Introduction

*If I can’t picture it, I can’t understand it.*

—ALBERT EINSTEIN

Safety-Critical Systems (SCS) are systems whose failure can lead to personal injury/death or to environmental damage (Knight, 2002). Many of these systems must be certified by a designated authority prior to initial deployment. Certification of these systems as safe is accomplished by examining the design and safety engineering activities such as safety analysis and by examining the artifacts produced by those activities. SCS are examined rigorously through precise safety engineering techniques, which contribute with artifacts that are combined with the system’s specification to build an assurance case for the system’s certification.

Software Product Lines (SPL) has proven to be a successful and emerging approach to develop software products (Pohl et al., 2005). It has been widely used in the development of SCS in several domains, such as automotive (Thiel and Hein, 2002) and avionics (Bergey et al., 2005); however, there are several issues with reusing artifacts from one system in another (Liu et al., 2011). SPL, providing answers to some of these issues, is an integrated approach in which artifacts generated in an asset development process can be effectively reused in a product development process when the assets are constrained to a very specific domain (Clements and Northrop, 2002). In this dissertation we present an approach on conducting safety analysis in a product line of SCS.

Safety analysis techniques for Safety-Critical Software Product Lines (SCSPL) will require analysis techniques that scale to the huge number of interaction combinations among the assets of a product line. Due to the direct impact of these systems and the need to assure that the development process has not introduced risk, it is necessary to analyze
1.1. MOTIVATION

This research comes at a critical time since SCS are increasing in complexity and becoming more integral to our everyday lives. Many SCS are also Cyber-Physical Systems (CPS), systems where software controls hardware elements, have real-time requirements and impacts humans directly (Lewis and P., 2010), which makes the verification even harder.

Applying product line development techniques to SCS offers the advantage of rapidly building a set of products using the customizability of SPL; however, this process must not introduce risks through either the system’s composition with various components or through the interactions between components’ ports of communication through which the components interact (Braga et al., 2012a).

The architectural model proposed is well defined and provides the ability to model large-scale architectures in a single analyzable model that can be incrementally refined (Feiler and Gluch, 2012).

1.2 Problem Statement

This work investigates the issues in composing safety critical components into a family of safety critical systems, and provides an architectural approach to conducting safety analysis in a Safety-Critical Software Product Lines (SCSPL) context to reduce the complexity of managing safety in those types of systems.

1.3 Statement of the Contributions

As a result of the work presented in this dissertation, the following contributions can be highlighted:

- A study of Safety-Critical Software Product Lines (SCSPL), which can provide
the research and professional community an overview of the state-of-the-art in the field including the gaps found that could be the basis for new research issues.

- **A product line architecture model**, a design for a family of cruise control systems using Architecture Analysis and Design Language (AADL), which provides opportunities for addressing reuse due to its strong typing and precise syntax / semantics for modeling performance-critical systems.

- **An approach to conduct safety analysis from an architecture perspective** which combines validation/verification using AADL, safety requirements created using the System-Theoretic Process Analysis (STPA), and design patterns all applied to the model.

In addition to the contributions mentioned, papers about safety domain, making use of the AADL architecture model developed for this work, were accepted for publications / presentations:

   Title: Architecture-Led Pedagogical Artifacts as a Unifying Theme

   Title: An Architecture-centric Approach for Safety Critical Software Product Lines

   Title: Analysis and Design of Safety-critical, Cyber-Physical Systems

The full project can be found at the Git Repository1.

### 1.4 Research Design

The Exploratory Research method (Library, 2016) was chosen to guide this work for several reasons. It gives us a well grounded picture of the field, tells us whether the study is feasible or not, and helps us develop more refined issues to be analyzed in a greater degree and thus, generate new research questions.

---

1[https://github.com/rose2s/AADL/tree/master/CACC model](https://github.com/rose2s/AADL/tree/master/CACC model)
1.5 RELATED WORK

Since exploratory research gives us a direction for future research and new techniques to develop, this work gathers enough information to position graduate researches in safety-critical software product lines domain.

The research process for this study was defined in five steps:

1. Building a product line architecture model on the following Safety Critical Systems: Cruise Control (CC), Adaptive Cruise Control (ACC), and Collaborating Adaptive Cruise Control (CACC) using AADL (Feiler et al., 2006);

2. Conducting safety analyses by using the traditional System-Theoretic Process Analysis (STPA) (Lutz et al., 1998) to identify potential hazards and safety requirements;

3. Illustrating the types of assets in the asset base for a Safety-Critical Software Product Lines (SCSPL). Two design patterns from (Douglass, 2003) are designed in AADL and discussed in terms of how one can replace the other in a system depending upon the product desired.

4. Using the error model framework - Error Modeling Annex (version 2) (EMV2) (Feiler and Gluch, 2012) to propose an alternative safety analysis technique based on building an error model and addressing the hazards and causes that were generated by the safety analysis methods in step 2.

5. Discussing differences between the STPA and the error model safety analysis techniques including the starting points of the algorithms and the completeness of the approach.

1.5 Related Work

In the last few years, several research efforts have been devoted to the definition of approaches related to AADL. The following studies were considered related because of having similar ideas to this work.

• Delange and Feiler (2014)
  Delange and Feiler, in their work *Architecture fault modeling with the AADL error-model annex*, used AADL and its Error Model Annex to support architecture fault modeling and automated safety analysis. They used a dual redundant flight
guidance system and discussed the automation of different safety analysis methods presented in the SAE ARP4761\textsuperscript{2} standard, emphasizing the benefits of automation.

- **Procter and Hatcliff (2014)**
  In their work, *An Architecturally-Integrated, Systems-Based Hazard Analysis for Medical Applications*, Procter and Hatcliff tailored the System-Theoretic Process Analysis (STPA) to the domain of Medical Application Platforms (MAP apps). They built an AADL-based language and tooling for the semi-formal modeling of MAP app architectures to provide a proof-of-concept tool that aids the transition between design and analysis. Their tool uses AADL’s error modeling annex as output for a proposed report format.

These related works have presented an architecture fault modeling approach including fault propagation ontology, which they applied to a dual redundant flight guidance system (Delange and Feiler, 2014) and to the domain of Medical Application (Procter and Hatcliff, 2014); whereas we have presented an architecture fault modeling approach, applied it to safety-critical system, and applied the approach not in a single system, but a family of Cruise Control systems. For that reason, we have illustrated safety critical design patterns as an example asset base for fault tolerance, along with AADL error modeling, to manage safety in Safety-Critical Software Product Lines (SCSPL).

### 1.6 Dissertation Structure

The remainder of this dissertation is organized as follows:

- **Chapter 2** reviews the necessary topics needed to understand the remainder of this work: Safety-Critical Systems (SCS), Software Product Lines (SPL), Safety-Critical Software Product Lines (SCSPL), Hazard Analysis techniques, and Architecture Analysis and Design Language (AADL).

- **Chapter 3** describes a running case design to evaluate our architectural approach, a Cruise Control SPL. Also explains main concepts about AADL, the design language used to structure this work.

- **Chapter 4** describes System-Theoretic Process Analysis (STPA), the safety analysis technique used to identify potential hazards in order to support our proposal, and applies STPA to our running example.

\textsuperscript{2}Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems
• Chapter 5 addresses the importance of using design patterns in modeling safety-critical systems, and applies The Protected Single Channel Pattern (PSCP) and Heterogeneous Redundancy Pattern (HRP) to our model.

• Chapter 6 describes issues in composing safety critical components into a safety critical application, conducts safety analysis of safety-critical products using an architecture-centric approach through the Error Modeling Annex (version 2) (EMV2) in AADL, and discusses the issues of managing the safety analysis of product lines.

• Chapter 7 summarizes the main contributions, limitations of the approach, outcomes and directions for future work.
2 Background

*Have the courage to follow your heart and intuition.*
—STEVE JOBS

2.1 Introduction

This Chapter describes background information relevant to understand the remainder of this work. It is organized as follows: Section 2.2 introduces concepts regarding Safety-Critical Systems (SCS); Software Product Lines (SPL) is introduced in Section 2.3; Section 2.4, Safety-Critical Software Product Lines (SCSPL), sketches the relationship between the previously addressed concepts; Section 2.5 presents three traditional hazard analysis techniques used in SCS; Section 2.6 introduces Architecture Analysis and Design Language (AADL); Chapter Summary is addressed in Section 2.7.

2.2 Safety-Critical Systems (SCS)

Safety-Critical Systems (SCS) are systems in which the safety of humans is connected to the correct operation of the system and, consequently, a system defect can lead to loss of life (Nair et al., 2014). For this reason, many SCS require safety certification and all require safety analysis. Analyzing the safety of a system involves collecting enough information that, for a specific type of failure, the system can be modified so that the effects of a defect are attenuated or so the chance that the system fails is below an acceptable level.

SCS are based on industry safety standards such as DO-178C (DO-178C/ED-12C, 2012) and ISO26262 (DISRV, 2009), aviation and automotive safety analysis techniques...
respectively. These safety standards provide a reference for defining evidence procedures to show that safety criteria are addressed appropriately \cite{Nair2014}. In some cases, safety criteria are set for particular organizations such as NASA's\footnote{National Aeronautics and Space Administration (NASA)} Software Safety Standard and the FAA's\footnote{Federal Aviation Administration (FAA)} System Safety Handbook \cite{Wong2014}. These standards recognize differing levels of risk to safety. Techniques such as integrity levels are used to evaluate the level of risk and the standards then describe the evidence required to guarantee safety. The standards do not require specific development processes rather they specify outcomes that assure quality.

### 2.3 Software Product Lines (SPL)

Software Product Lines (SPL) is a development methodology in which a family of products are built from a common set of assets. The architecture of the family has variation points, pre-identified locations where engineers can select from a set of possible features that can be composed into that variation point, and units of behavior designed to fit in the variation point \cite{VanGurp2001}. These selections result in a product that is more specific to the needs of the customer, and by sharing a common base across a family of products rather than expending more time developing each product individually from scratch, the cost of development and the time needed for developing each product is decreased while the quality of the family of products is increased \cite{Clements2012}.

The planning required for a software product line is more extensive than for a single copy program. This has led to an emphasis on architecture and detailed specification, which is useful for safety critical systems. Matinlassi \cite{Matinlassi2004} compares architecture design methods for software product lines.

### 2.4 Safety-Critical Software Product Lines (SCSPL)

Safety-Critical Software Product Lines (SCSPL) are software product lines in which each product is safety critical, but constructed using the extensive reuse techniques of a product line. Safety-Critical Software Product Lines (SCSPL) have been created in diverse areas such as medical infusion pumps \cite{Procter2014} and unmanned aircraft \cite{Braga2012}. Safety-Critical Software Product Lines (SCSPL) must meet
the expectations of its safety properties in all available variations, which makes reuse management and safety guarantees major issues in a safety analysis (Jing Liu and Lutz, 2007).

Strengths of SCSPL are the strong support for model-based development and the ability to combine safety check lists with commonality / variability analysis (Lutz et al., 1998). Therefore, conducting hazard analysis along with verification activities at the design level would allow us to discover faults early enough to design recovery strategies before implementation.

A major issue for safety critical product lines is the integrity of the reuse process (Braga et al., 2012b). Safety critical software can cost as much as $1,000 per line of code. Reusing code is certainly cost justified, but that process must be carefully managed. The detailed error modeling, discussed later in this paper, is one way of assuring that components being integrated are a match. Procter and Hatcliff (2014) illustrates the use of error modeling as a means of assuring appropriate reuse. Inheritance is a second technique in that the family of components intended to be interchangeable is managed as a single inheritance hierarchy.

## 2.5 Existing Hazard Analysis Techniques

From a safety perspective, formally defining requirements is a part of the product line development process and is an integral part of conducting a safety analysis (Lutz et al., 1998). Since this process is common to both, the results are used by both processes. Ensuring that safety analysis is focused solely on the commonalities, and is thus applicable to all products is difficult, and processes for performing this composition are just now materializing.

The key to safety assurance is the identification and mitigation of hazards. To control hazards, it is necessary to identify them. We briefly describe three of the most common hazard analysis techniques in SPL: Software Fault Tree Analysis (SFTA), Software Failure Mode, Effects and Analysis (SFMEA), and System-Theoretic Process Analysis (STPA).

### 2.5.1 Software Fault Tree Analysis (SFTA)

SFTA is a safety analysis technique which helps to find causes of undesired events using a top-down approach and takes advantage of boolean logic to drive a backward search (Dehlinger and Lutz, 2004). The root node represents a hazard, a set of conditions
that can lead to an accident, and its child nodes represent the causes of the hazard (Lutz et al., 1998). The output of the process is a list of potential hazards that the system can experience.

### 2.5.2 Software Failure Mode, Effects and Analysis (SFMEA)

SFMEA is a bottom-up technique, which has been used to provide a systematic form of failure analysis to improve reliability (Jing Liu and Lutz, 2007). It starts with the fault of a single component progressing to an overview of the whole system (Dehlinger and Lutz, 2004). This technique lists the set of components that comprise the system and their related failure modes. Then, for each component, an analysis is performed based on the consequences of their failure modes on both themselves and the overall system.

### 2.5.3 System-Theoretic Process Analysis (STPA)

Unlike SFTA and SFMEA, which focus on component failures, STPA takes the interaction of components into consideration when determining which faults are possible (Dulac and Leveson, 2005). A guiding principle behind STPA is that safety is not just a failure problem, it is also a control problem. It does not just find failures, it changes the operation of the system through constraints by providing safety requirements during system design (Wagner and Abdulkhaleq, 2015).

### 2.6 Architecture Analysis and Design Language (AADL)

AADL is an architecture-centric model-based engineering approach, developed in 2004 by the Society of Automotive Engineers (SAE) (Feiler et al., 2016), which contributes to the verification and validation of safety-critical systems (Feiler and Gluch, 2012). AADL is strongly typed and its precise semantics make it an ideal choice for representing safety-critical domains.

Traditional safety analysis can be integrated with AADL by making use of the tool set in the Open Source AADL Tool Environment (OSATE). One of the tools, Reqspec, is a requirement specification language that links the requirements directly to components in an AADL model. These results in improved traceability ensure consistency between requirements and architecture. AADL, through the Error Model Annex Version 2 (EMV2) property set and error annex, supports the design of detailed error models that allow the
explicit modeling of hazard parameters and explicit design for error propagation from one component to another (Feiler et al., 2016).

2.7 Chapter Summary

In this chapter, we addressed important concepts to this work: Safety-Critical Systems (SCS), the area of Software Product Lines (SPL), and the two areas integrated (safety-critical of product lines) including definitions, strengths, and challenges. We also presented existing Analysis techniques for SCS such as SFTA, SFMEA, and STPA. Finally, we introduced the design language used in this work: AADL. More Details of AADL will be explained in the next chapter.
3.1 Introduction

In this chapter, we characterize the three types of products we work on: Cruise Control (CC), Adaptive Cruise Control (ACC), Collaborating Adaptive Cruise Control (CACC) in section 3.2, and present its product map. Section 3.3 shows the graphical view of Cruise Control SPL model. The Assume Guarantee REasoning Environment (AGREE), a AADL model checker, is presented in Section 3.4, and Section 3.5 explains Error Modeling Annex (version 2) (EMV2), AADL subset language used in our approach.

3.2 Overview of the example problem

In order to conduct safety analysis with System-Theoretic Process Analysis (STPA) technique (described in Chapter 4), introduce the concepts of AADL’s Error Annex language, model the system architecture, and describe our approach (Chapter 6), we run an example problem of a safety-critical product line on the automotive domain.

As a motivation for this example, over the past 10 years, fatalities on the roadways of the United States have decreased by 25 percent (NHT, 2016). In the same period of time, the American Automobile Association (AAA) Foundation for Traffic Safety pointed out that some safety technologies have contributed to this trend (Mehler et al., 2014).

Next we present definitions and characteristics of a Cruise Control product line.

3.2.1 Cruise Control (CC)

CC assists in maintaining a set speed without keeping the foot on accelerator. It will disengage as soon as the driver hit the brake pedal, and it will not engage at speeds less
3.2. OVERVIEW OF THE EXAMPLE PROBLEM

than 25 mph (40 kph) (See Appendix A, Section A.1). The system uses a speed sensor as the basis for accelerate/decelerate decisions.

3.2.2 Adaptive Cruise Control (ACC)

ACC is an extension of the basic Cruise Control, which has the additional ability of detecting target vehicles using a forward-looking object detection sensor, and measuring some attributes such as distance and relative velocity (Widmann et al., 2000). Unlike CC, which can only be set to a specific speed, ACC can automatically adjust speed when another vehicle, in the same lane, changes its speed in order to maintain a pre-set distance between the vehicles.

If the vehicle monitored by the sensor slows down, the module sends a signal to the braking system to decelerate. When the sensor no longer detects an object or detects a widening gap, the system accelerates the automobile back to the set speed. One benefit of this type of control system is avoiding rear end collisions where the front of the vehicle strikes the rear of a lead vehicle (Mehler et al., 2014).

3.2.3 Collaborating Adaptive Cruise Control (CACC)

Collaborating Adaptive Cruise Control (CACC) extends the CC by adding a vehicle-to-vehicle (V2V), communication mechanism to an ACC system to allow cars to “collaborate” by communicating with each other (Shladover et al., 2015). V2V communication is an integral part of a system design known as the Internet of Things (IoT), research into which is being funded by the United States Department of Transportation (DoT), and the National Highway Traffic Safety Administration (NHTSA) (Van Arem et al., 2006). The result is that cars can travel in close groups, referred to as platoons, safely with braking and accelerating done cooperatively.

In Table 3.1, we can see four different types of products that we can build from this family of products. In addition to 4 products differentiated by features, the product line offers 2 different packages for CACC products differentiated by safety levels. Package one uses a non-redundant set of hardware and a simple software analysis pipeline. Package two uses redundant hardware and two, independently designed and developed, software analysis pipelines which are interconnected. The design patterns for these are discussed in Chapter 5.
3.3 Architecture Analysis and Design Language (AADL)

AADL has both a graphical and textual syntax which can be instantiated to an instance model that describe a system’s components and the connections between them. Figure 3.1 shows the high level graphical notation of cruise control SPL.

As we can see in Figure 3.1, the model has four sensors (radar, camera, GPS, speedometer); a sensor handler for each device; a logger component which keeps information about the sensors; the vehicle controller where the system makes the decisions, sends feedback to the Interface controller, which sends the feedback to the user interface component; vehicle controller also receives speed information from the speed controller, which sends command to the brake and throttle actuators.

### Table 3.1 Product Map

<table>
<thead>
<tr>
<th>Features</th>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1 CC</td>
</tr>
<tr>
<td>Buttons</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Platoning</td>
<td>x</td>
</tr>
<tr>
<td>Sensors</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Temperature</td>
<td>x</td>
</tr>
</tbody>
</table>

3.4 Assume Guarantee REasoning Environment (AGREE)

AGREE is a compositional verification tool for AADL based on the widely-used assume-guarantee contract verification method AADL models (Murugesan et al., 2013). Designers state their assumptions about input and specify guarantees concerning output if the assumptions are met. Note that AGREE splits the assume-guarantee contracts from their behavior specification. The assume-guarantee contract is placed in the component specifi-
3.4. Assume Guarantee REasoning Environment (AGREE)

Figure 3.1 A graphical view of an AADL component
3.5. ERROR MODELING ANNEX

cation and the behavior specification is placed in the component's implementation. This way, the multiple implementations common in a SPL can use the same assume-guarantees while having their own behavior specification. An agree clause example, based on our running example, is shown in Figure 3.2.

![Figure 3.2 Excerpt of AGREE statement of User Interface Component](image)

For example, the code at lines 3-5 of the AGREE excerpt assumes the system will *set speed up/down* when cruise control is enabled, i.e., *turn_on_off* is true. Similarly, the code at lines 6-12 assumes the system will *level up/down, set a gap distance, and pause/resume* when *turn_on_off* is true.

3.5 Error Modeling Annex

One reason that AADL is a fitting language for describing SCS is because it has language annexes that extend its modeling capacity to support non-architectural system aspects. The error modeling annex, abbreviated to EMV2 in its second version, extends AADL to permit the design of error types, sources, propagations, behavior, etc.

EMV2 consists of special properties in AADL that can be connected to component types. They are high-level libraries which can contain error types, failure state machines, and other constructs (Procter, 2016). EMV2 has a library of common error types. Figure 3.3 has the hierarchy of timing related errors. For example, early and late data delivery are types of item timing errors (see Figure 6.1).
3.6 Chapter Summary

This chapter described our running case design, a Cruise Control SPL, to evaluate our architectural approach; explained main concepts about Architecture Analysis and Design Language (AADL), the design language used to structure this work, and AGREE, its model checker to support verification activities.

Figure 3.3 Timing related errors in the EMV2 error type hierarchy (Procter et al., 2015)
4 Conducting a Hazard Analysis Using STPA

4.1 Introduction

This chapter describes the process of applying a hazard analysis using System-Theoretic Process Analysis (STPA) and applies it to a Cruise Control SPL. Section 4.2 gives a hazard definition; Section 4.3 describes STPA, the safety analysis technique used to identify potential hazards in order to support our proposal; Finally, it generates a STPA report with the results of the analysis. Major steps of STPA, defined in Section 4.3.1, are the identification of systems boundaries, accident levels, accidents, hazards, safety constraints, control actions, and process model.

4.2 Hazard Definition

According to the United States Department of Defense (DoD, 2000), a hazard is any real or potential condition that can cause injury, illness, or death to personnel; damage to or loss of a system, equipment or property; or damage to the environment.

As we can see in Figure 4.1, the hazard control process involves identification of hazards, assessment of hazard risk, and the control and verification of hazards and associated risks (Ericson et al., 2015). This process defines a closed-loop within which hazards are tracked and mitigation are refined until acceptable actions are verified.
4.3 Description of STPA

System-Theoretic Process Analysis (STPA) works by creating scenarios that could lead to an accident such as problems arising from unsafe and unintended interactions between system components. These scenarios are created through a systematic process providing guidance for identifying the potential for inadequate control of the system leading to a hazard.

STPA is a broadly-applicable hazard analysis technique, and so it is suitable for a range of safety-critical systems (Leveson, 2011). Procter and Hatcliff (2014) proposed in his dissertation a form of STPA report using major concepts in STPA based on Leveson’s ideas. He stated that the format presented provides guidance to analysts, and produces a more uniform final product. The STPA-based report format is divided into three sections, background, STPA fundamentals, and unsafe control actions and their causes, explained in more detail in Section 4.3.1. We used this proposed report approach to apply STPA to our running scenario in the automotive domain.

4.3.1 STPA Report Concepts

These concepts were extracted from Leveson (2011).

Background

The Background section of the modified STPA report consists of a scenario that contextualizes and explains the need for the system being analyzed;
STPA fundamentals

The STPA fundamentals section contains the following:

- System Boundaries
  Which domain concepts and behaviors are in the system and which are not;

- Accident Levels
  The accident levels are degrees of severity of the types of accidents that can happen due to system operation. Levels are useful when tradeoffs among goals are required in the design process. The levels also help us to determine which conditions are part of the hazard and which are part of the environment;

- Accidents
  An unplanned event that results in a loss of human life or human injury, property or environmental damage;

- Hazards
  A system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident (loss);

- Safety Constraints
  Requirements related to preventing the hazards from occurring and ensuring safe operation;

- Control Actions
  All of the actions taken by the control system are control actions. The inputs and outputs of the system’s components typically include information on their range of possible values, which determine the flows - nominal and error. The process model determines whether the inputs lead to unsafe actions;

- Process Model
  The system maintains an internal model of the state of the real world and the processes that affect that state. This process model is used to reason about operation and to make decisions about how to respond to control actions.

Unsafe Control Actions

This section describes the unsafe control actions that are possible in the system as designed. An unsafe control action is a control action that could be provided in a manner
or at a time that is unsafe. It is identified by reasoning about the effects of each control action and whether or not the action could be performed in an unsafe way. This reasoning is driven by a number of guide-words that Leveson provides (Leveson (2011), page 218), listed in Table 4.1. The guide-words are used to identify scenarios that have the properties denoted by the guide-word.

- **Causes and Compensations**
  The analysis determines the causes of the unsafe control actions and how the system compensates for these actions. For example, the scenarios could violate the component safety constraints.

## 4.4 Analyzing the Cruise Control SPL using STPA

Now we have seen the STPA fundamental concepts, we apply them to the Cruise Control SPL.

### 4.4.1 Background

![Figure 4.2](Image)

**Figure 4.2** The New York Times | Florida Traffic Crash Report

**Context** "A person was killed driving a car with Adaptive Cruise Control (ACC) enabled. There was a technical failure of the automatic braking system, and for our
specific scenarios we will assume that happened for following the reasons: the radar sensor sent late data to the process model or the sensor sent no data to the controller component”.

**Tesla Accident**

Figure 4.2 shows a representation of a real fatal accident where a Tesla car, using Autopilot system, crashed into a truck that was turning left in front of it, on May 7, 2016 in Florida, US. The vehicle ran under the truck’s trailer and continued off the road, hitting a power pole before coming to a stop (Singhvi and Russell, 2016).

According to the National Highway Traffic Safety Administration (NHTSA), the vehicle mistook the truck with a road sign (Yadron and Tynan, 2016). Questions have arisen about the safety of the car’s crash-avoidance system, and after the fatal accident, the company made some upgrades to its autopilot system. One important upgrade was the onboard radar, which before was only a supplementary sensor to the primary camera (Tesla, 2016). This fatality is still part of a preliminary evaluation of Tesla’s Autopilot system by the NHTSA.

**Assumptions**

The system operates under one of three modes - Cruise Control, Adaptive Cruise Control, or Collaborating Adaptive Cruise Control. The system transitions from CACC mode to ACC mode when there is no platoon with which to communicate. The system will transition from ACC mode to CC mode when there is no gap to maintain.

**4.4.2 Preliminaries**

**System Boundaries**

The cruise control system is essentially a state machine that is a subsystem of the vehicle. Variables that affect the state of the acceleration/deceleration of the vehicle are within the both the subsystem and system boundaries and those that are not within the subsystem boundary may be within the outer boundary or outside it. The sensors that measure speed and distance and the values they produce are within the system boundary. Intrusion sensors and other values not related to speed are outside the system.

**Accident Levels**

Safety analysis is only interested in those levels of accidents that threaten the safety of humans. The levels of accident related to vehicles include:
4.4. ANALYZING THE CRUISE CONTROL SPL USING STPA

L1. Death or serious injury to a human
L2. Damage to the car

Accidents

Any system operates in an environment where unintended events can occur and the outcome of that event can be one of the accident levels. Accidents involving a cruise control can result in:
A1. Humans are killed or injured [L1];
A2. Cars crash into each other resulting in property damage [L2];
A3. Car crash into a non-car [L2].

Hazards

The CC systems can cause harm to humans in the following ways:
H1. Component receives late data from the sensor and takes action too late to be effective [A1] or [A2] or [A3]
H2. Sensor sends no data to the controller and no action is taken to avoid the accident [A1]
H3. Car drives on a rough road and the radar sensor sends OutOfRange data [A1]
H4. ACC estimates of distance and speed of vehicle ahead are incorrect
H5. ACC violates the safe distance between the ACC vehicle and the vehicle in front

Safety Constraints

The system must be constrained in ways that avoid the hazards:
C1. System must inform the display of sensor information at a rate that is proportional to the speed of the vehicle [H2]
C2. System must alert about road conditions that affect acceleration and deceleration [H1, H3]
C3. Radar sensor must detect objects that are in the path of the vehicle, and send data to the controller in time for the system to respond [H1].
C4. The module must take action to compensate when no data is available [H2].
C5. The module must propagate hardware errors to the software [H4].
4.4. ANALYZING THE CRUISE CONTROL SPL USING STPA

**Control Actions**

1. radar_sensor:sensor_data_out -> radar_info = obstacle location (2D coordinate pair), velocity (2D vector)
2. speed_sensor: sensor_data_out -> velocity (float vector)
3. vehicle_controller:speed_controller:cmd -> String ("increase", "decrease", "off", "on");
4. vehicle_controller:UI_feedback -> Boolean

**Process Model**

The process model for a Adaptive Cruise Control (ACC) system behaves the same as the Cruise Control (CC) system when there is no vehicle in front. It is a simple state machine (see Appendix A, section A.3) in which the set state depend on its previous condition and on the provided inputs.

The process model for ACC is shown in Figure 4.3, a feedback control loop. The controller is assigned requirements to enforce on the controlled process, which it does by issuing control actions to change the state of the controlled process. For controllers in a safety control structure, the assigned requirements must ensure that the safety constraints are handled (Leveson and Thomas, 2013).

![Figure 4.3 The Process Model for ACC Module](image)

In a cruise control system, for example, the system sets and maintains a speed until
told to stop. If the speed falls below the target it instructs the vehicle to speed up. When the system is following another vehicle the state machine is slightly more complicated. It maintains a distance gap between the two vehicles by speeding up or slowing down the vehicle to match the vehicle in front.

Information about the controlled process state (feedback) is provided by sensors and control actions are implemented on the controlled process by actuators.

### 4.4.3 Unsafe Control Actions

<table>
<thead>
<tr>
<th>Control Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radar_sensor:sensor_data_out</td>
<td>Radar sensor does not provide the relative speed and distance of objects ahead of vehicle</td>
</tr>
<tr>
<td>Not Providing</td>
<td>Radar sensor provides incorrect data for the target vehicle speed</td>
</tr>
<tr>
<td>Providing Incorrectly</td>
<td>Radar sensor is applied too long that the ACC module does not get the relative data signal of the target vehicle</td>
</tr>
<tr>
<td>Applied too long</td>
<td>Radar sensor is stopped too soon that the ACC module does not get the relative data signal of the target vehicle</td>
</tr>
<tr>
<td>Stopped too soon</td>
<td>The data comes too early when the distance to a forward vehicle is too far</td>
</tr>
<tr>
<td>Early</td>
<td>The data comes too late when the distance to a forward vehicle is too close</td>
</tr>
<tr>
<td>Late</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.1** Excerpt from Radar Unsafe Control Actions

*Causes and Compensations*

- Providing Input Incorrectly: Radar sensor provides incorrect data
- Cause: Incorrect values are gathered from the sensors
- Compensation: Reusing the last data that was sent and/or interpolate the last set of data.

### 4.5 Chapter Summary

In this chapter, it was presented a hazard analysis process with STPA. It presented the STPA report concepts extracted from Leveson (2011), and applied to safety scenarios. Next chapter how Safety Critical Design Patterns were used to support our proposal.
5.1 Introduction

In this chapter, we present the start of an asset base for an SCSPL. Two design patterns from Douglass (2003) are described that are widely used in safety critical systems for fault tolerance. The patterns are described using the Gang of Four template (Gamma et al., 1995). We show an AADL model of each of the patterns and discuss how they work within the Safety-Critical Software Product Lines (SCSPL).

5.2 Motivation

The motivation for applying design patterns for fault tolerance was the Tesla fatal accident described in Section 4.4. In the accident, the autopilot didn’t apply the brake because the camera sensor confused a truck for a road sign; In other words, the sensor confused a moving obstacle for a stationary object, and the system makes different actions for each scenarios.

Actually, one of the main challenges of using sensors for a braking system is how to prevent false positives, in which a car might think an overhead highway sign, for example, is an obstacle to be avoided. One way of preventing this is having redundancy in the system. Redundancy in hardware and software is explained in the next section.

5.3 Redundancy in hardware and software

Hardware components experience static failures based on design faults and dynamic failures due to physical faults such as friction and voltage surges. Once deployed the
majority of software does not change significantly; therefore, any failures are due to
design errors.

Having two identical hardware components decreases the probability of failure dra-
matically since the non-deterministic effects of wear and tear give the two components
differing life times. Unless two software components are created independently, i.e.
separate development teams, their life times will be approximately identical.

In our product line a vehicle may have a single sensor for noticing obstacles and a
single actuator for carrying out commands or the basic design may be supplemented by
a backup sensor and actuator. A channel uses the sensor input to compute the actuator
commands for a single sensor. A modified channel asset is created by combining two
channels and adding logic that enables switching between channels.

5.4 Applying the patterns

The adaptive cruise control system adjusts the speed of the vehicle based on the location
and speed of obstacles that are in the path of the vehicle. The basic model includes a
radar sensor and a camera sensor that identify objects such as the vehicle in front of the
sensing vehicle. There is also a GPS sensor to provide information about the changes in
location of the sensing vehicle.

5.4.1 Pattern Description

There are several styles for presenting patterns in the abstract. This section uses the
Gang of Four style (Gamma et al., 1995) with the Implementation section being given
in AADL since it is the architectural aspects that are of interest. The Participants and
Collaborators sections are combined into one section. Each of the two patterns will be
presented by dividing the narrative into these sections.

• Intent
• Motivation
• Applicability
• Collaborators
• Consequences
• Implementation
5.4. APPL YING THE PATTERNS

• Related Patterns

5.4.2 Protected Single Channel Pattern (PSCP)

Below is the The Protected Single Channel Pattern (PSCP) description.

**Intent:** Provide some protection for less critical elements without the expense of full redundancy.

**Motivation:** Having redundancy is expensive in terms of hardware replication and development costs. In a Safety-Critical Product Line, not all products will require the most expensive redundancy. For that reason, the PSCP is a low cost configuration that works with a single sensor, single actuator.

**Applicability:** PSCP can be used anywhere it is desirable to have increased security and reliability, but it is acceptable to tradeoff some of that increased assurance for a less expensive implementation.

**Collaborators:** The pattern encompasses three types of elements that collaborate to provide an end to end design.

![Figure 5.1 Single Design Pattern Example](image)

*Sensor* - The Sensor collects data to support decision making.
5.4. APPLYING THE PATTERNS

Channel - The channel is the main computational structure that houses the participants. Within the channel is a set of interacting components that provide the business logic for computing the actions needed from the actuator.

Actuator - The Actuator carries out the actions computed by the pattern.

DataTransformation (dt) - A component that contains safety and reliability policies. Multiple of these components can be chained together serially to compute complex actions.

DataValidation (dv) - A component that continually runs checks on the software and hardware. Techniques such as Class Responsibility Collaboration (CRC) checks are used to look for errors in computation.

InputProcessing (ip) - A component that collects input from the Sensor and performs initial processing such as grouping inputs into meaningful packets.

OutputProcessing (op) - A component that acts as the driver for the Actuator. Computed values are translated into commands to the Actuator.

Consequences: The pattern clearly delineates the various elements used in the steps to compute the actuator commands. If a new type of actuator is deployed, the software changes should be limited to the Output Processing component. New sensors require changes to the InputProcessing component. Changes in algorithms for sensor fusion belong in the DataTransformation component. New validation rules impact the DataValidation component.

Implementation: Figure 5.1 shows the three main components in this pattern: sensor, channel, and actuator. It also shows the internal structure of the channel, which is composed of components that handle input and output processing, internal data transformation, and data checks (Douglass, 2003). The validity check might use a CRC approach or some other technique for recognizing data corruption. PSCP is a low cost approach because it can only recognize faults that are due to singular events.

Related Patterns: In particular, the Heterogeneous Redundancy Pattern, which is presented next, includes a higher degree of actual redundancy.

5.4.3 Heterogeneous Redundancy Pattern (HRP)

Below is the Heterogeneous Redundancy Pattern (HRP) description.

Intent: The HRP provides redundant channels as an architectural means to improve safety and reliability beyond the levels offered by PSCP.

Motivation: Unlike Single Channel, the Heterogeneous Redundancy Pattern uses multiple channels, as shown in Figure 5.2, with independent designs and/or implemen-
Applying the Patterns to improve the detection of faults including systematic faults. This is the most expensive kind of redundancy because not only is the recurring cost increased, but development cost is increased due to the redundant design and implementation costs (Douglass, 2003). However, the close relationship of the pattern to PSCP can reduce the implementation cost in a product line organization.

**Applicability:** HRP is applicable for highly critical systems that justify the additional expense for increased safety and reliability.

**Collaborators:** The pattern encompasses the same three types of elements that collaborate in PSCP to provide an end to end design.

- **Sensor** - The Sensor collects data to support decision making.
- **Channel** - The channel is the main computational structure that houses the participants. Within the channel is a set of interacting components that provide the business logic for computing the actions needed from the actuator.
- **Actuator** - The Actuator carries out the actions computed by the pattern.

Each channel has the same internal elements as in the PSCP pattern with one additional component, shown in Figure 5.3.

- **DataTransformation** - A component that contains safety and reliability policies. Multiple of these components can be chained together serially to compute complex actions.
- **DataValidation** - A component that continually runs checks on the software and hardware. Techniques such as CRC checks are used to look for errors in computation.
- **ActuationValidation** - A component that checks the appropriateness of the commands about to be sent to the actuator. It must consider the time lag due to the computation and checking.
- **InputProcessing** - A component that collects input from the Sensor and performs initial processing such as grouping inputs into meaningful packets.
- **OutputProcessing** - A component that acts as the driver for the Actuator. Computed values are translated into commands to the Actuator.

**Consequences:** The pattern clearly delineates the various elements used in the steps to compute the actuator commands. If a new type of actuator is deployed the software changes should be limited to the Output Processing component. New sensors require changes to the InputProcessing component. Changes in algorithms for sensor fusion belong in the DataTransformation component. New validation rules impact the DataValidation component.

The HRP is more expensive than single redundant patterns since the components in
5.4. APPLYING THE PATTERNS

Figure 5.2 Dual Channel Pattern Level 1

Figure 5.3 Dual Channel Pattern Level 2
each channel must be developed by an independent team using independent techniques. HRP is less reliable than a triply redundant pattern. The designer must evaluate the cost/reliability tradeoff in the context if their application.

**Implementation:** The components in the separate channels are independently developed and then deployed. There may even be different algorithms. The two channels are interconnected to facilitate corrections.

The Data Validation component again uses a technique such as CRC, but error correction may flow between the two channels in both directions.

**Related Patterns:** Much of the structure of HRP is inherited from the PSCP.

### 5.5 Product Line Architecture

The product line architecture shows the possible configurations of assets to make products. Our purpose in choosing these two patterns was to illustrate that there are similarities in redundancy approaches that make the design of assets relatively easy. The similarity of the channels encapsulates the changes that must be made to whichever cruise control configuration is selected.

### 5.6 Chapter Summary

In this chapter, we have presented two patterns that enhance the safety and reliability of a system. They represent assets appropriate to the asset base of a Safety-Critical Software Product Lines (SCSPL). The pattern descriptions make the major tradeoffs visible and provide information for decision making.
6.1 Introduction

In this chapter, we propose an approach to safety analysis that uses the error modeling capabilities of the Error Modeling Annex (version 2) (EMV2) to AADL. It is made by illustrating architecture fault modeling with EMV2 with the running example, a family of Cruise Control systems. These capabilities are described as they are needed for the safety analysis technique.

6.2 Issues in composing safety critical components into a SCSPL

Products in a product line are composed of reusable assets, but traditional views of safety are defined in terms of complete products. Building a safety critical product from a set of product line assets requires some proof that hazards are being handled. We think of each reusable asset as potentially contributing to a system level hazard and we will term each asset that actually is part of a hazard an asset contributor.

Issue 1 - A significant issue with ensuring safety in a software product line is assuring that all possible errors are handled appropriately. Design of error handling flows must initially be applied at the primitive asset level. Complete error modeling involves component error flows within the component and error propagations that signal incoming and outgoing error flows.

Issue 2 - A second significant issue is showing that all system level hazards are
6.3. A DESCRIPTION OF THE SAFETY ANALYSIS TECHNIQUE

handled. Eventually the product instance is completely assembled. Safety design requires that all error flows that do not terminate in a sink be examined as a potential hazard.

Issue 3 - A third significant issue is properly aligning the boundaries of the reusable assets with those boundaries of the regions defined for various portions of the system of interest. The “system” has multiple boundaries depending on the questions to be analyzed. For example, a control loop forms a system for the purposes of examining control actions. A larger scope would include the inputs and outputs to and from the control loop within the automated system and the constraints in those inputs and outputs due to the particular product. Finally, a still larger scope would include the externals to the product such as humans who interact with the product.

6.3 A Description of the Safety Analysis Technique

In this section we layout the tasks in the safety analysis technique. They are roughly in order; however, these are not discrete steps where one step is finished before the next one starts. The result of one task may result in the need to repeat a previous task.

6.3.1 Basic component error models

Unlike STPA, our safety analysis technique starts at the bottom of the system implementation hierarchy. A component error model is created within the design of each foundational component. Figure 6.1 shows, at lines 19-28, a component error behavior declaration associating the Failed state with an outgoing propagation.

Identify basic sources of error behavior

The computation in each component has the potential to encounter errors. The errors may come from other components or may come from logic problems within the component. The error annex of AADL provides a state machine syntax to describe the component behavior in the presence of errors. The state machine outputs commands which determine the actions of the component, see Lines 24 - 27 in Figure 6.1.

Cover the Error Ontology

To be certain that the error model is as complete as early design information allows, the error ontology, defined in the AADL’s EMV2 Annex language, provides a comprehensive, if not complete, set of error types, see Figure 6.2. As the error model is built the designer
6.3. A DESCRIPTION OF THE SAFETY ANALYSIS TECHNIQUE

Figure 6.1 Component Error Behavior

examines each error type hierarchy to determine if the errors defined in that hierarchy are possible in the current context. If it is, then that error type is added to the error model, see Figure 6.1, Lines 5 - 15. The ontology is systematically searched until every error type is either rejected or included in the model.

Figure 6.2 Error Ontology Hierarchy (Feiler, 2011)
6.3. A DESCRIPTION OF THE SAFETY ANALYSIS TECHNIQUE

Identify Sinks and Propagations

Error flows, within a single component, that terminate in an error sink are considered to be handled and are eliminated from the safety analysis. Error flows that propagate errors out of ports are possible contributors to system-level hazards. As we can see in Figure 6.3, each sensor is a potential source of NoData errors due to a reading failure, declared as an error source for the outgoing error propagation NoValue on sensor_data_out. It is an "error source" since it is the component where the error originates.

```
1. abstract generic_sensor
2.  features
3.    sensor_data_out: out data port;
4.  flows
5.    sensor_source : flow source sensor_data_out
6.    properties
7.      latency => 1 ms .. 3 ms applies to sensor_source
8.      SEI::PowerBudget => 5.0W;
9.  annex EMV2 (**
10.    use types error_library;
11.    end propagations;
12.    flows
13.      sensor_data_out : out propagation (NoValue);
14.      ef0 : error source sensor_data_out (NoValue);
15.      end propagations;
16.    properties
17.      emv2::hazards => 
18.        ((Failure => 'NoValue',
19.          description => "No data from the sensor"),
20.        ));
21.      applies to sensor_data_out.noValue;
22.    **);
23. end generic_sensor;
```

**Figure 6.3** Error propagation for Generic Sensor

generic_sensor_handler is both a source of NoData errors due to failure and passes on incoming NoData errors from the generic_sensor as indicated by the error path declaration. In addition, generic_sensor_handler shows an incoming propagation of BadValue and LateValue errors. This accommodates potential data corruption in the communication between generic_sensor and generic_sensor_handler. These specifications are shown on the right side of Figure 6.3.

Select Propagating Error Flows

Some of the error flows will only exist within a single component and will not contribute to a system hazard. We select only those flows that propagate errors to associated components. See sensor_data_out on Lines 13 - 17, left side of Figure 6.3 and sensor_data_in on Lines 18 - 30, right side of Figure 6.3. In the example, sensor_data_out out data port
6.3. A DESCRIPTION OF THE SAFETY ANALYSIS TECHNIQUE

of generic_sensor component propagates NoValue error to sensor_data_in in data port of generic_sensor_handler component.

6.3.2 Composite Error Models

The purpose of the compositional error behavior specification is to define a mapping of error states of sub-components of a system into error states of the system itself (Delange and Feiler, 2014).

Integrate Component Error Models

When multiple components are composed inside another, the component error models of the composed components are integrated into a composite error model. The encapsulating component also has a component error model to be taken into consideration (see Figure 6.4). For the Cruise Control System, we have defined a separate error state machine for each mode (CC, ACC, CACC) as well as transitions among the modes.

```
1. annex EMM2 [**
2.  use types error_library;
3.  use behavior error_library::stateMachine;
4.  
5.  composite error behavior
6.  
7.  states
8.  [radar_handler.Failed and camera_handler.Failed
9.    and gps_handler.Failed and speedometer_handler.Failed
10.  ] -> Failed;
11.  [radar_handler.Failed and camera_handler.Failed] -> Failed;
12.  [radar_handler.Failed or camera_handler.Failed] -> Operational;
13.  [radar_handler.Operational and camera_handler.Operational
14.    and gps_handler.Operational and speedometer_handler.Operational
15.  ] -> Operational;
16.  **);
```

**Figure 6.4** Compositional Error Behavior for ACC

In the excerpt in Figure 6.4, we specify a compositional error behavior mapping for ACC where the Operational state is when all sensor components are Operational. Another possible Operational state in ACC mode can be reached whether radar_handler or camera_handler are Operational; otherwise, the system must be in Failed state. The logical expressions used in that mapping effectively represent the logic in a fault tree intended to compensate for failures in system components.

Safe asset composition requires that data, control, and error flows between the two assets be connected and that the connected flows must be type compatible. Part of design for safety requires that the design handle hazard contributors.
6.3. A DESCRIPTION OF THE SAFETY ANALYSIS TECHNIQUE

Specialize Errors

The component error models usually will each have some of the same errors, ones that are related to the nature of the domain. If the same error comes into the composite from two different components they may have to be handled in two different ways. These need to be identified and different subtypes of the common error type created. These can then be included in the composite error model.

Identify Sinks and Propagations

Identifying sinks and propagations in a composite component is similar to what it was in the foundational components. For errors that are handled in the composing component, i.e. flow into sinks, these errors do not contribute directly to a system hazard and are not included in the analysis. However, that sink may result in other errors being generated. Others are propagated to the next level.

Integrate Error Models

The state machine in the composite error model integrates the actions of the component error models. These may eventually propagate errors to the system level.

6.3.3 Design top level error behavior

The component/composite hierarchy of error models is repeated until the top of the system implementation hierarchy is reached. Errors are propagated into this top level and new errors may arise from the top level computation. At this point, errors that would be propagated to a next level, which does not exist, can cause failures. The question to ask is whether these failures have the potential to harm humans.

Finally, we examine each end-to-end flow that flows to the system level. As we can see in Figure 6.5, a full path is traced from an error source to an error sink. At each sink we attempt to determine whether the error is actually handled.

Figure 6.5 Radar end-to-end Flow
6.3. A DESCRIPTION OF THE SAFETY ANALYSIS TECHNIQUE

Figure 6.6 shows the end-to-end error propagation of the error type \textit{LateValue} from radar sensor as an error source to \textit{brake_actuator} as an error sink leading to a hazard such as \textit{H1}: "Component receives late data from the sensor and takes action too late to be effective" and the accident \textit{A1}: "Humans are killed or injured" (see Section 4.4.2).

As error flows are traced from source to sink most will be handled by error sinks at...
some level in the implementation hierarchy. The error flows that are not handled in a sink pose a potential hazard to successful execution. Each of these are traced back through the implementation hierarchy and identified as hazard contributors.

Figure 6.5 and Figure 6.7 are the textual view of the model of the system, its sub-components including the devices radar_sensor, radar_handler, and the process vehicle_controller. As we already declared the error propagation of in/out ports of each components, here we just need to declare the connections between these component interfaces. The connections radar_handler_conn and radar_conn bind the sensor and the controller together.

The hazard report can be produced by tracing back along the error flows to see the interactions at all levels of the implementation hierarchy. Any unhandled errors are recorded. At the system level, all error sinks are examined to determine whether that error may not be adequately handled.

### 6.4 Discussion of the technique

The error modeling supports the discovery of hazards by looking at error flows in the error model of the full system. Through error propagations we can follow from error sink to error sources, where hazards, if they exist, would originate as hazard contributors. This approach works from the lowest levels of the implementation to the highest as opposed to STPA, explained in Chapter 4, where the analysis must work with the complete system.

#### 6.4.1 Benefits of Using the Error Model Annex

Using Error Modeling for hazard identification clearly carries a lot of benefits to the system. First of all, AADL is open-source, so it can be used by anyone, and it has been used by many researchers in Safety-critical systems (Knight, 2002).

EMV2 allows us to perform the safety analysis compositionally starting with end-to-end flows within each component (as shown in Figure 6.1) rather than using solely traditional safety analysis such as STPA, which only works with system level hazards.

Since it allows the tracing of every error flow (error propagation), software engineers have detailed coverage for each component. Furthermore, applying the error ontology we identify all possible incoming and outgoing propagations, so that if we have that in the model, it will be very straight forward to analyze the composed system at the next level.

Finally, conducting safety analysis at the design level also contributes to early identification of potential problems, including single points of failure, the effects of multiple
failures, and unexpected failures (Delange and Feiler, 2014); frequent re-analysis as the architectural design evolves, and extension of safety analysis into the software system architecture.

### 6.4.2 Limitations of Using the Error Model Annex

The validity of our approach is based on the validity of the error ontology. The ontology has been used by numerous experts in embedded systems and it has been used in papers that have been refereed by experts. Although there has been no formal validation study of the ontology we believe there has been sufficient experiential validation.

### 6.5 Chapter Summary

In this chapter we have presented several of the tasks needed for conducting a safety analysis using an error analysis. The technique addresses three issues related to safety critical product lines. 1) The safety analysis follows the implementation hierarchy in the architecture and uses the component/composite decomposition to modularize the analysis. 2) By starting at the lowest level of the implementation hierarchy and searching the error ontology the model is more likely to be comprehensive. 3) By creating a component error model for each component the model is portable and can be reused in many products.
Conclusion

Safety is a fundamental quality attribute so it should be considered early in domain engineering while designing the SPL architecture. Achieving safe reuse of product line assets requires being aware of the interactions among design assets. AADL’s ability to model large-scale architectures from many perspectives in a single analyzable model complements product-line development methodologies.

This work investigated the issues in composing safety critical components into a family of Cruise Control systems, and provided an architectural approach to conducting safety analysis in a Safety-Critical Software Product Lines (SCSPL) context to reduce the complexity of managing safety in those types of systems.

The proposed approach was to use the strong type checking of AADL and the modeling / constraint checking provided by AADL’s annexes along with the tools given to define interfaces and information flows. The Error Annex enabled us to trace error propagations and find potential hazards earlier during design. One strength of this approach is that there is no need for extensive domain expertise to achieve good and complete safety requirements for safety critical systems.

A second strength of our approach is the use of the error ontology defined in the error annex for different error types. This gives the less experienced analyst a bounded search space to ensure that they consider all possible error types as sources of error flows.

7.1 Lessons Learned

An important lesson we have learned throughout this study is that safety will be well handled by integrating approaches instead of relying on a single technique. On the path to guarantee safety in a safety critical system, we needed to apply/integrate different strategies. System-Theoretic Process Analysis (STPA), was very useful to identify
potential hazards in order to support our proposal, but since it only can be applied to the system level, the bottom-up approach of Error Modeling Annex (version 2) (EMV2) could work together to iteratively drive forward not only the hazard analysis but the system design itself.

7.2 Future Work

A pilot project was developed and analysed in this work. However, we are aware that the architecture model is not completed. Besides, a case study and/or evaluation makes necessary to validate our approach. Thus, some important aspects that our future work will investigate are described as follows:

- refinements of existing product line development processes to support the design of safety critical systems;

- additional examples and comparison of our approach to other safety analysis techniques; and

- the ALISA set of domain specific languages - Reqspe, Verify, and Assure - which augment the AADL model and what they can contribute to the development of safety-critical systems. Reqspe is a language for capturing requirements. Verify is a language for writing verification actions such as static analyses and test cases and linking them to mechanisms for automatically executing those actions. Assure is a language for defining assurance cases in which verification activities are used to build evidence for some type of certification process. There are examples of the use of these languages in coordination with the cruise control AADL model in the same github repository.

We have developed an AADL model that gives us a sandbox for further study of safety critical systems.


Appendix
Cruise Control Safety Requirements

This appendix presents more details about the features of Cruise Control System addressed in Chapter 3. Section A.1 describe some safety requirements needed to design the product line. Section A.2 describes the requirements of Cruise Control SPL designed in the Requirements Definition and Analysis Language (RADL). Section A.3 shows the state machine for CC, ACC, and CACC.

A.1 Cruise Control (CC)

- CC won’t engage at speeds less than 25 mph (40 kph)
- CC will disengage as soon as driver hits the brake pedal
- CC will disengage if the vehicle speed decreases more than 10 mph (16 km/h) below driver’s set speed while driving uphill.
- Set Speed will range between 30 - 102 mph
- The set speed will change in approximately 1 mph (2 km/h) increments.

A.2 Requirements Definition and Analysis Language (RADL)

RADL is the requirement definition and analysis tool environment supported by AADL and others architecture description languages. RADL has analysis and traceability capabilities, making verification of requirements by the design. Figure A.1 show most of cruise control requirements.
A.3 State Machine

Figure A.1 RADL Model

Figure A.2 CC State Machine
A.3. STATE MACHINE

Figure A.3 ACC State Machine

Figure A.4 CACC State Machine