CARNEGIE MELLON UNIVERSITY

CARNEGIE INSTITUTE OF TECHNOLOGY

SYSTEM SAFETY AS AN EMERGENT PROPERTY IN COMPOSITE SYSTEMS

Jennifer Ann Black

Pittsburgh, Pennsylvania

April 20, 2009

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS

for the degree of

DOCTOR OF PHILOSOPHY

in the department of

ELECTRICAL AND COMPUTER ENGINEERING

Advisor: Philip Koopman

Department Head: T. E. Schlesinger

Dean: Pradeep K. Khosla

"Ah," said Arthur, "this is obviously some strange usage of the word 'safe' that I wasn't previously aware of."

- Douglas Adams, *The Hitchhiker's Guide to the Galaxy*

Abstract

Correctly specifying requirements for composite systems is essential to system safety. In a distributed development environment, safety requirements must be clearly defined for subsystems. Unfortunately, decomposing non-functional requirements, also known as goals, is not always straightforward. Quantifiable goals, such as cost or performance, may be decomposed by allocating a fixed limit on each component. However, system safety is usually not expressible as a sum of parts. Rather, it is considered to be *emergent*.

This thesis defines emergent and composable behaviors in the context of formally specified goals, and identifies useful special cases in which emergent system goals may be partially composable. Indirect Control Path Analysis (ICPA) is introduced as a new technique for identifying and documenting safety goals for components, using control flow and goal coverage strategies to guide goal elaboration.

ICPA was applied to a semi-autonomous automotive system from a commercial automotive research laboratory and the goals and subgoals were monitored at run-time in a partial implementation of the vehicle in a simulation environment. Violations of both the goals and subgoals identified several critical design defects in the incomplete implementation. In some situations, false positive detection at the subsystem level identified problems in the subsystems that were masked by redundant goal coverage. False negative detection at the subsystem level in some of the scenarios suggests the set of subsystem safety goals only partially composes the system-level behavior. The results demonstrate proof of concept of the ICPA technique for defining system safety subgoals in a real system.

Acknowledgments

This thesis is dedicated to my husband Eric, whose love and encouragement sustained me through my struggles. Thank you for realizing this pilgrimage with me. I also thank my Mom, Dad, and family, both immediate and extended, for their love, strength, and endurance. You are the standard-bearers who guide my advance.

I thank my academic advisor Professor Philip Koopman for his guidance, for his collaboration, and for making this thesis possible. You have given me the opportunity to begin, the tools to proceed, and the motivation to finish.

I thank Dr. Alan Baum for his advice and support as a committee member, a supervisor during my internship at General Motors, and a friend. I also thank my committee members Professor Dan Siewiorek and Professor John Knight for their time, feedback, and patience.

I thank Beth Latronico, Justin Ray, Susan Farrington, and Suzie Laurich-McIntyre for lending their ears and shoulders from time to time. I also thank the researchers at GM R&D and Honeywell Labs who shared their expertise and provided feedback on my research. In particular, I thank Tom Fuhrman, Max Osella, Doug Rheaume, Shengbing Jiang, Mira Supal, Larry Peruski, Michael Paulitsch, and Brendan Hall.

Finally, I thank my research sponsors for funding this endeavor. This work has been supported by the General Motors-Carnegie Mellon University Vehicular Information Technology Collaborative Research Lab (GM-CMU VIT CRL), a National Science Foundation Graduate Research Fellowship (NSF GRF), the United States Council for Automotive Research (USCAR), a General Motors Fellowship, the Pennsylvania Infrastructure Technology Alliance (PITA), Honeywell Labs, Bosch RTC, and Bombardier Transportation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Research has shown that system safety, defined by Leveson as "freedom from accidents or losses" [50], is closely linked to the quality of requirements. One case study of safety-critical software isolation revealed that some dependencies between safety-critical and non-safety-critical components were missed during safety analysis [4]. Other research has traced safety-related software errors and operational anomalies in spacecraft to latent requirements, misunderstood requirements, and misunderstood interfaces between the physical system and the software [59][60].

System safety is difficult to define and ensure in a distributed development environment. In one study of software errors in spacecraft [59], miscommunication between development teams was the primary cause of safety-related interface faults. In other industries with less regulation or less centralized oversight of development, the problem may be worse. For example, in the automotive industry vehicle subsystems are often developed by different internal departments or external suppliers. These suppliers may not have access to requirements for the vehicle as a whole, or for other subsystems. Similarly, vehicle manufacturers may have little control over the development process or internal details of components purchased from vendors. In order to manage safety at the subsystem level prior to system integration, system safety requirements must be clearly defined for subsystems.

Unfortunately, decomposing non-functional requirements, also known as goals [18] or

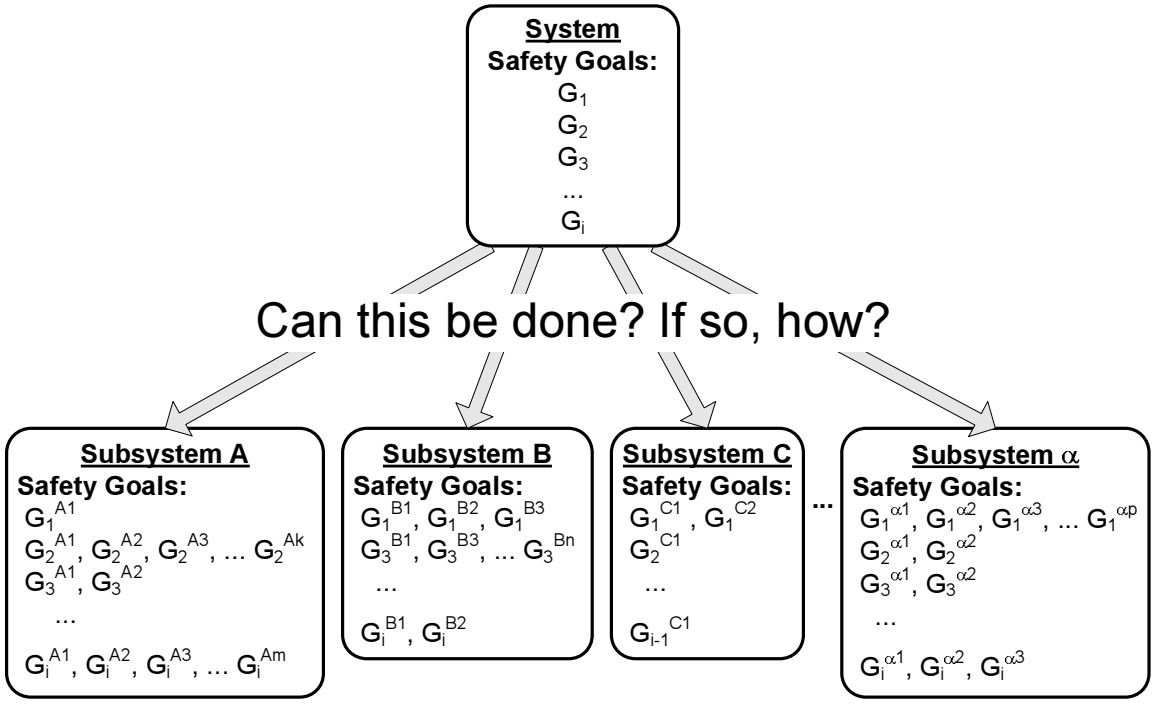


Figure 1.1. Safety goal elaboration from system to subsystems

quality attributes [39][7], is not straightforward. Some quantitative goals, such as cost or performance, may be decomposed by allocating a fixed limit on each component in the functional decomposition [65]. However, other goals may be qualitative or not easily represented as a sum of parts. For example, an automotive safety goal might be "the vehicle shall experience zero collisions." Unlike performance goals, where the concept of "time" is the same for systems and subsystems, the concept of "collision" in system safety does not have the same meaning at different levels of the system hierarchy.

1.1 Problem and scope

This thesis addresses the problem of defining safety goals for subsystems in a composite system, which is illustrated in Figure 1.1, using the notation described in [41]. For each system safety goal G_i there is some set of subgoals $\{G_i^{\alpha k}\}$, k = 1, 2, ..., j, where α is the

subsystem to which the subgoal is assigned. The primary research questions are:

- What does it mean for a goal to be composable or emergent?
- When is it possible to decompose a system-level goal?
- Is partial decomposition possible, and if so, is it useful?
- How can full or partial decomposition of system safety goals be achieved?
- How do we evaluate the value of a partial decomposition?

The problem scope is defined with respect to requirements engineering, hazard analysis, and system decomposition:

Requirements engineering. Any system development process, safety-critical or not, begins with requirements elicitation. Goal Oriented Requirements Engineering (GORE) [18] is one approach to specifying system requirements from high-level system goals. In GORE, goals are first refined into subgoals that can be assigned to individual agents (subsystems, software components, the user, etc.), and then operationalized into functional requirements (the behaviors that realize the goal). In general, the objective of goal elaboration is to find a set of subgoals and operationalized requirements that satisfy the parent goal. With system safety goals, the objective of goal elaboration is to analyze all agents that might impact the safety goal and identify a set of subgoals and operationalized requirements for some of those agents that satisfy the parent goal and also satisfy a goal coverage strategy. This thesis proposes additional tactics for safety goal elaboration in the GORE framework.

Hazard analysis. In safety-critical system design, different hazard analysis techniques are used throughout the development process to identify system hazards and their causes, and to define requirements to avoid the hazards by eliminating their causes, or to control the

consequences of the hazards. Preliminary Hazard Analysis (PHA) may be used early in requirements development to identify hazards related to general system functionality [50]. For example, a PHA in an automotive system might be used to identify hazards related to acceleration, deceleration, steering, and driver distractions. As the system design progresses, Fault Tree Analysis (FTA) may be used to trace top-level hazards to the underlying events that cause them [50]. An FTA of an unexpected deceleration hazard in an automotive system might trace the hazard to a fault in object detection. The objective of the techniques proposed in this thesis is not to identify hazards, root causes of hazards, or system safety goals for avoiding the hazards. Rather, it is to identify subsystem safety goals that satisfy the system goals that have been obtained from other hazard analysis techniques.

System decomposition. Decomposition is a common strategy for managing system complexity that is applied to three different types of design attributes: structures, behaviors, and goals [41]. In most systems, decomposition is applied to structures, behaviors, or some combination of both. For example, an automotive system is decomposed by both behavior (cruise control, electronic stability control, anti-lock braking, etc.) and structure (brakes, transmission, steering, etc.). This thesis addresses the problem of decomposing system safety goals along an existing structural + behavioral decomposition.

1.2 Indirect Control Path Analysis

This thesis proposes and evaluates a safety goal elaboration technique called *Indirect Control Path Analysis (ICPA)* [11][12]. ICPA is a top-down analysis approach, similar to FTA. Whereas FTA traces a top-level hazard to its lower-level causal events, ICPA traces a top-level state variable in a system safety goal through the design to the agents that influence it. ICPA uses a table structure similar to Failure Modes and Effects Analysis (FMEA) [50]

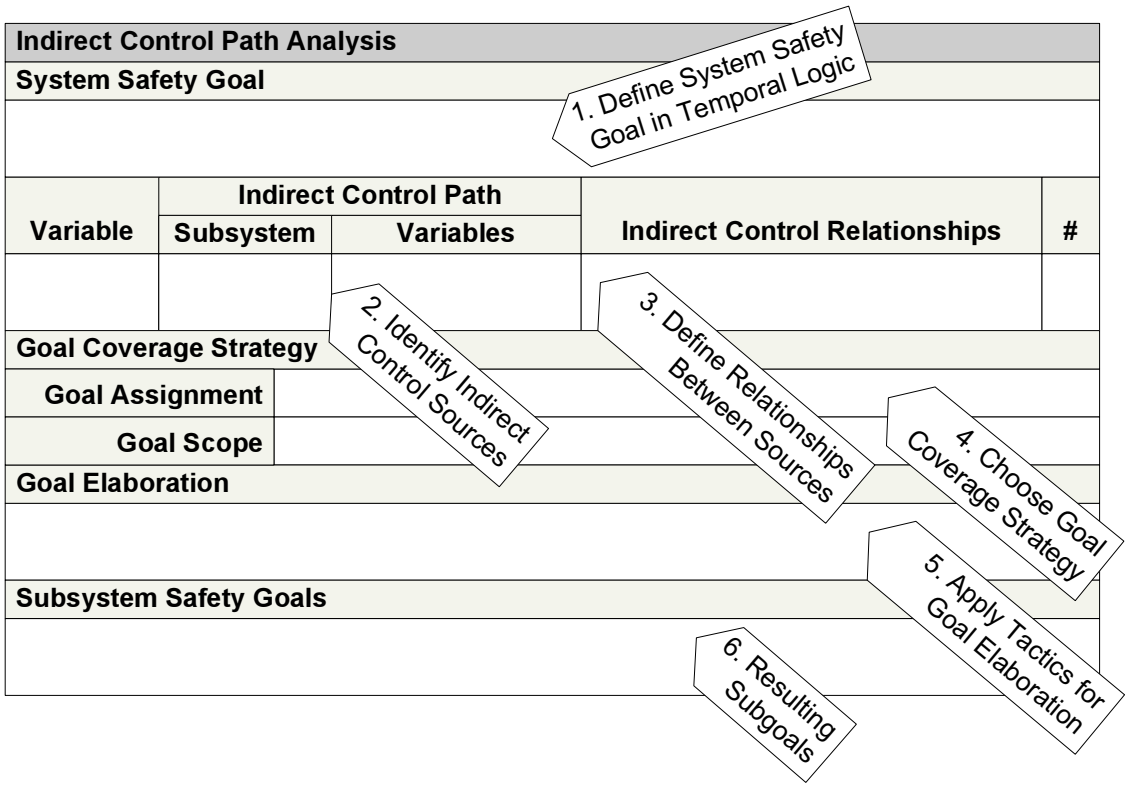


Figure 1.2. ICPA table layout and steps

to record these indirect control relationships. Figure 1.2 shows the layout and steps of an ICPA. Agents that indirectly influence a state variable belong to the indirect control path and require further analysis of their relationships to the root variable.

A *goal coverage strategy* is a plan for allocating subgoals to ensure that a high-level goal is met. Each strategy is defined by *goal assignment* and *goal scope*. Goal assignment defines which indirect control sources have subgoals and how those subgoals relate to each other (e.g., single responsibility, redundant responsibility, and coordinated responsibility). Goal assignment may be driven by physical limitations of the system (e.g., actuation delays) or by possible loss of monitorability and controllability by agents in the system. Goal scope defines how closely the safety subgoals satisfy the system safety goal. Although it may be possible for agents to satisfy the original safety goal exactly, it may sometimes be necessary

or desirable to assign subgoals that are more restrictive than the original safety goal. The result of an ICPA is both a set of subsystem safety goals that satisfy the parent goal, and a record of the critical assumptions, goal realizability tactics, and goal coverage strategies used to define them.

1.3 Thesis contributions

This thesis makes the following contributions:

- **I provide a formal definition of emergence within a framework of goal decomposition.** In GORE and ICPA, system safety goals are represented as temporal logic expressions. This thesis defines *emergent*, *fully composable*, and *emergent but partially composable* goals within this mathematical framework, both with and without redundancy, and identifies useful special cases in which partial decomposition of emergent safety goals is possible.

- **I create a technique for guiding system safety goal elaboration in a directed and documented way.** Indirect Control Path Analysis (ICPA) is a technique for decomposing system safety goals from the system level to the subsystem level. Decomposition is guided by architectural control flow and formal representation of system goals and indirect control paths. It provides a structured, deliberate, and documented analysis of potential subgoal agents and goal coverage strategies. It also allows demonstration by mathematical proof or model-checking that the set of subsystem goals produced with ICPA partially or fully compose the system goals under a specified set of critical assumptions.

- **I demonstrate that monitoring of system safety goals and subgoals can detect some hazards at run-time.** In this thesis, ICPA is applied to a semi-autonomous

automotive system from a commercial automotive research lab. The safety goals for the system and key subsystems are monitored at run-time in a simulation-based implementation of the vehicle. Results demonstrate that monitoring of safety goals at the subsystem level can detect system safety-related errors. In addition, monitoring of safety goals at both the system and subsystem level is valuable for identifying certain safety-related errors that may be imperceptible to system testers or the driver.

The remainder of this thesis is organized as follows. Chapter 2 presents background information and related work. Emergence is defined in the context of formally specified goals for composite systems in Chapter 3. ICPA is introduced in Chapter 4. In Chapter 5, ICPA is evaluated in a simulation-based implementation of a semi-autonomous automotive system. A discussion of those results is found in Chapter 6. Chapter 7 presents conclusions and future work.

Chapter 2

Background and Related Work

2.1 Overview

This chapter describes the background work related to the ICPA technique for elaborating system safety goals in composite systems. Section 2.2 describes background information on system safety and hazard analysis techniques related to the work presented in this thesis. Section 2.3 presents background on structuring, eliciting, and refining requirements, as well as related approaches to defining and elaborating system safety goals. Section 2.4 describes the problem of emergence in composite systems. Finally, Section 2.5 describes related work in run-time monitoring of system safety properties.

2.2 System safety

The application domain for this thesis is safety-critical distributed embedded systems. An *embedded system* is a system that includes computer control but is not used for general-purpose computing. An embedded system is *distributed* if control is divided among sub-systems or components, rather than centralized in one dedicated control unit. Within the scope of this thesis, the term *safety* refers to system safety, which Leveson defines as "freedom from accidents or losses" [50]. A safety-critical system is one that can directly or indirectly contribute to a loss event. The definition of a loss is dependent on the domain,

and can be very broad (loss of life, bodily injury, loss of system, financial cost, etc.). Some examples of safety-critical distributed embedded systems are cars, aircraft, and elevators.

Hazards are system states in which an accident could eventually occur [50]. The combination of the likelihood that a hazard will occur and its consequences is *risk* [80]. The goal of safety engineering is to reduce risk, either by reducing the likelihood that the hazard will occur, or by reducing its consequences. Although in most systems it may not be possible to eliminate all hazards, it is still important to try to reduce risk as much as possible.

Hazards can arise when faults manifest as errors in the system. Avizienis et al. define an *error* as a deviation in correct system state, and a *fault* as its root cause [6]. A fault could be in hardware (e.g. a sensor that fails 'ON') or software (e.g. buffer overflow). Sometimes the fault is in the requirements, design, or implementation. Some research has shown that critical software errors result more often from defects in the requirements and misunderstandings about the software/system interface than from implementation defects (e.g., source code bugs) [59][60].

Safety is considered a non-functional system property. Non-functional properties, also known as goals [18] or quality attributes [39][7], are not specific actions the system must perform, but rather are characteristics the system should exhibit while doing whatever it does. Other examples of non-functional system properties are dependability, performance, cost, and maintainability. The focus of safety is hazard prevention; thus, *safety requirements* often take the form of constraints on system behavior.

A common misconception about safety engineering is the belief that designing a system to be reliable will also make it safe. Although safety and reliability are related, they are not equivalent. Whereas the goal of a reliable system is continuation of correct service, the goal of a safe system is prevention of accidents or losses. For example, if the throttle of an automobile fails such that the throttle is closed and the vehicle rolls to a stop, the system

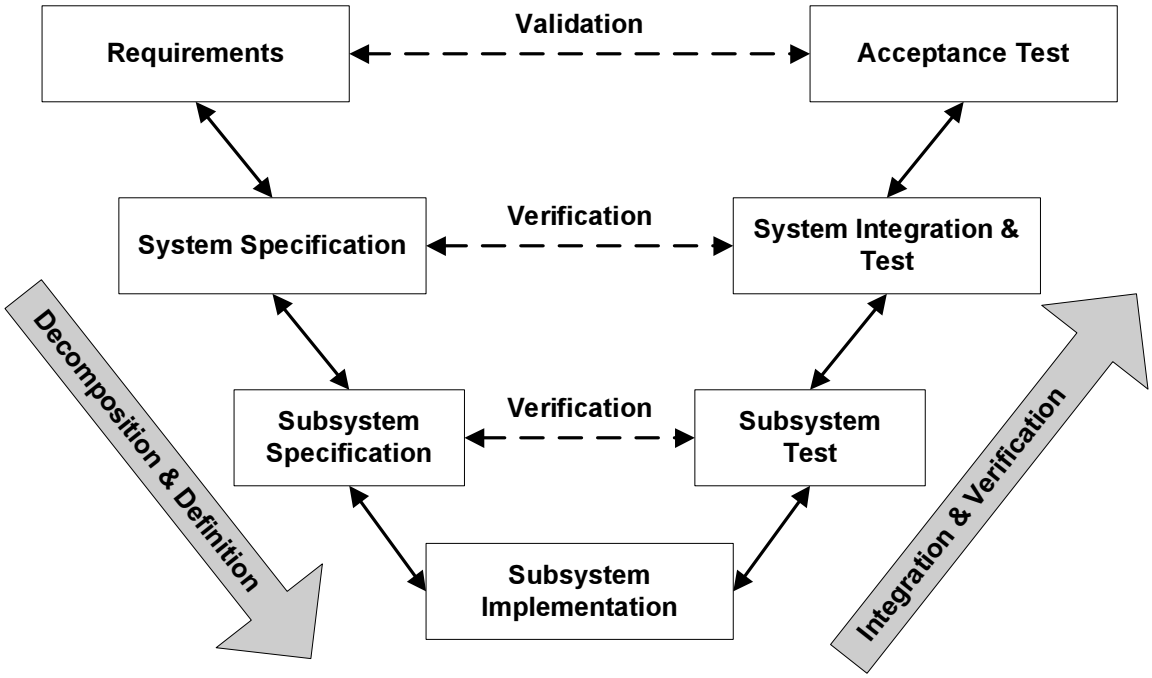


Figure 2.1. V-model of system development (adapted from [29])

might remain safe, even though it has lost service and is not reliable. Likewise, a system that is reliable might not be safe for a given set of environmental operating conditions (e.g., a vehicle that is correctly driving down the road with another vehicle in its path).

2.2.1 Hazard analysis

Hazard analysis provides a structured method of reasoning about hazard causation throughout the system development life cycle. One common representation of this cycle is the "Vee" model developed by NASA for the Software Management and Assurance Program (SMAP) [29]. A similar model of software development called the V-model was developed simultaneously by the European technology company IABG for the Federal Ministry of Defense of the Republic of Germany for use in military information technology systems [34]. Figure 2.1 shows a basic V-model adapted from the one presented in [29]. Design activities

on the left side of the V represent decomposition, from high-level system requirements to low-level component specifications. Activities along the right side of the V integrate components to form the complete system and verify and validate the implementation against the requirements and design specification. Traceability occurs between stages as the cycle progresses.

In the context of this V-model, hazard analysis begins during requirements specification and continues at each subsequent stage of development. In the early stages, hazard analysis is used to identify hazards, reason about their underlying causes, assess their risks, and develop measures to eliminate them, reduce their occurrence, or mitigate their consequences. Later, hazard analysis is used to verify that these measures have been implemented correctly. Traditional systems safety engineering relies on a variety of hazard analysis techniques, each with slightly different requirements and goals. This section describes four different hazard analysis techniques related to the work presented in this thesis.

Preliminary Hazard Analysis (PHA). Safety requirements elicitation typically begins with Preliminary Hazard Analysis (PHA) very early in development, during the requirements engineering stage [28]. The PHA produces a list of general hazards related to both the application domain and the particular application under review. It also usually includes some indication of hazard severity. As the development process and the system design progress, prevention and mitigation techniques are added for each hazard in the PHA. Because PHA occurs so early in development, it is generally a system-level analysis used to identify system-level safety goals. The ICPA technique presented in this thesis, which focuses on defining subsystem safety requirements from system safety requirements, does not replace the PHA, but complements it.

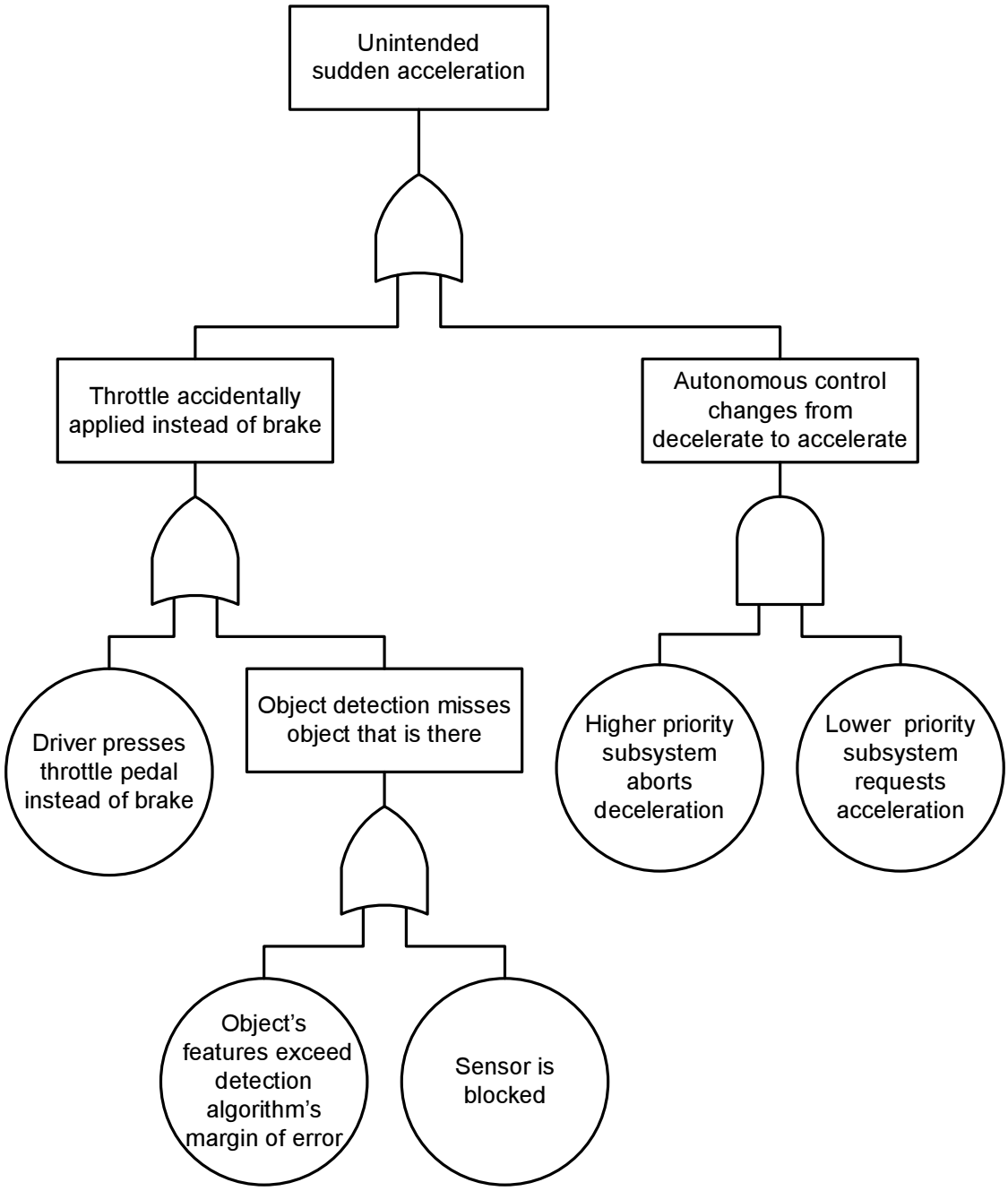


Figure 2.2. Partial fault tree for a semi-autonomous automotive system

Fault Tree Analysis (FTA). As the subsystem requirements and design specification is completed, other hazard analysis is used to identify component-level sources of system

hazards. FTA is a backward search technique that starts from a hazard and traces through its causal event chain to identify the fault or failure in the system that is the root cause [85]. Figure 2.2 shows a partial fault tree for a semi-autonomous automotive system. Component failure events are connected by logical AND and OR operators to indicate when an output event requires all or at least one input event. This allows failure scenarios to be constructed and simplified using Boolean algebra. In Figure 2.2, the hazard *Unintended sudden acceleration* could occur if a high-priority subsystem cancels an attempt to decelerate the vehicle at the same time as a low-priority subsystem requests a vehicle acceleration.

The goal of a traditional FTA is to identify and eliminate single-point failure scenarios, indicated by paths up the fault tree that traverse no AND gates. In the case of hardware failures, redundancy can be used to eliminate or mitigate the hazard. Although fault tree generation is manual, determination of hazard probability from component failure rates (if known) could be automated.

Software fault tree analysis (SFTA) [55] was the first significant attempt to adapt a traditional hazard analysis for software verification. This was also the first paper to define software safety as distinct from safety in traditional electromechanical systems. Other work in SFTA can be found in [32][63][22]. SFTA traces hazardous software outputs back through the code to the logic or input failures that could cause them. The results of the code-level SFTA are then used to determine where run-time assertions might be necessary. Leveson and Harvey provide templates for SFTA using symbols found in traditional FTA [55]. These templates demonstrate how common code structures, such as if-then-else and while loops, can be broken down into fault trees. Although code-level SFTA can be useful for identifying coding defects, such as not checking for infinite loop execution, it does not address faults in requirements that are common sources of hazards in software systems.

Component	Failure Modes	Causes	Probability	Effects
Long-range radar sensor	False positive	Signal noise	3×10^{-2}/hr	Could cause Collision Avoidance to randomly stop vehicle
	False negative	Signal noise	1×10^{-2}/hr	Could cause Collision Avoidance to miss an object

Figure 2.3. Partial FMEA for a semi-autonomous automotive system

Both FTA and ICPA are backward search techniques. The primary difference between FTA and ICPA is that fault trees trace hazards through the design to identify the underlying faults that cause them, whereas ICPA traces high-level safety goals through the design to define safety requirements for the subsystems and components. ICPA is concerned with defining goals for the functional behavior of components, rather than identifying faults that could cause those goals to be violated. A secondary difference is that FTA is recorded in a tree structure, whereas ICPA is recorded in tabular format.

Failure Modes and Effects Analysis (FMEA). FMEA is a forward search technique that lists potential faults in components and identifies their possible effects on the system [50]. This analysis is done manually by domain experts and produces a list of components, their failure types, and the resulting system effects in tabular form. A variant of FMEA is the Failure Modes, Effects, and Criticality Analysis (FMECA), which includes additional analysis of the criticality of each failure mode. Detailed instructions for performing a hardware FMECA can be found in [83]. Figure 2.3 shows a partial FMEA for a semi-autonomous automotive system that lists potential failure modes for a long-range radar sensor.

The goal of traditional FMEA is to identify components whose failure might lead to a hazard. Using this information, systems engineers can modify the design, usually by adding redundancy, to remove the hazard or reduce its consequences. One advantage of this type of hazard analysis is that the list of components, though possibly extensive, is usually known at the time of analysis. One possible disadvantage, particularly for software, is that all failure modes of those components might not be known.

Like traditional FMEA, Software Failure Modes and Effects Analysis (SFMEA) is a forward search technique that lists potential faults in software components and identifies their possible effects on the system [62][61][63][30]. Failure modes of software, however, are often more numerous and more complex than their hardware counterparts. Lutz and Woodhouse propose analyzing eight different software component failure modes [62]. Data failure modes represent faults in communication between components and include absent, incorrect, mistimed, or duplicate data. Event failure modes represent faults in software processes (programs in execution) and include abnormal process termination, process omission, incorrect logic, or mistimed events. For example, a critical mode flag (data) that suffers an incorrect value fault might cause the system to be set to an incorrect mode [62]. One that suffers a mistimed data fault might cause a delay in setting the system to the appropriate mode.

One key difference between SFMEA and SFTA is that SFMEA is performed on the requirements, rather than on the code. In their analysis of the Cassini spacecraft, Lutz and Woodhouse identified forty-eight issues in the requirements, twenty-five of which required changes to the requirements [62]. Four of the changes involved unresolved specification requirements and were linked to failure modes that had not previously been considered. It is important to note that this version of SFMEA only identifies which faulty software output behavior, with respect to its interaction with the rest of the system, is hazardous enough

to warrant more analysis; SFTA is still required to trace that faulty output behavior to its source (processor fault, sensor input fault, faulty software logic, etc.). In this capacity, SFMEA can be used to reduce the scope of SFTA, as Lutz and Woodhouse do in their analysis. It may also be possible to use SFMEA to trace faults in software inputs forward to hazardous software outputs, but this use of SFMEA is not explored.

The primary difference between FMEA and ICPA is that the former is forward search while the latter is backward. FMEA traces component failures through the system to the hazards they might produce, whereas ICPA traces system safety goals through the system architecture to define subsystem safety goals. One similarity between FMEA and ICPA is the use of tables for recording details of the analysis. Another is that both techniques are concerned with component I/O, with FMEA focusing on fault models of component I/O and ICPA focusing on the controllability and observability of system state by components.

STamP Analysis (STPA). Leveson introduced a new approach for accident reasoning using systems theory called Systems-Theoretic Accident Model and Process (STAMP) [52][54]. This new model is based on the assertion that traditional hazard analysis techniques do not work for software-intensive systems because software is complex and non-linear. Because most hazards in complex software-intensive systems are not caused by component faults, but rather by failures in requirements to adequately address safety, tracing the paths from hazards back to component failure and vice versa will not detect many common sources of hazards in software systems. In contrast, STAMP attributes hazards to control processes that fail to enforce safety constraints. This might occur because the control process itself is inadequate or inadequately executed, or because of inadequate feedback to the control process. For example, an inadequate control action might be a flaw in the control process for monitoring constraints or an overlooked hazard which was not assigned a safety constraint.

A STAMP hazard analysis (STPA) [53][25] begins with defining the control architecture

of the system. Once controllers and control flow are known, failure modes are identified for each component in the architecture. The four types of control failures are missing control, incorrect or unsafe control, late control, and premature stop of control [53]. These types of subsystem control failures are then traced to potential causes using a formal model of the system and control failures in the SpecTrm-RL language [14].

One tenet of the STAMP hazard model is that safety analysis of components cannot be separated from safety analysis of the system. Leveson asserts that although the system itself may be hierarchical, certain properties such as safety emerge only at the highest level of the hierarchy, and that individual components are neither safe nor unsafe outside the context of the system. In this view of safety, separating hazard analysis of any components from system-level analysis is not appropriate. However, the component control constraints generated during the STPA process are comparable to the concept of safety goals for subsystems in the ICPA framework. Similar to ICPA, the STPA analysis is based on the control architecture of the system. Unlike ICPA, which uses backward search from safety goals defined at the system level through the control architecture to define safety goals for components, STPA uses forward search from components to system hazards. The backward search in STPA is used to identify the source of component control failures.

PARCEL. Johnson introduced the Programmable electronics systems Analysis for Root Causes and Experience-based Learning (PARCEL) [38][37] hazard analysis technique for software-intensive systems. It traces accidents and incidents back to failures in the software development cycle. The main idea of PARCEL is that adverse events occur when development processes fail to meet safety-critical system development standards, such as IEC61508 [35] or DO-178B [73]. PARCEL utilizes flow charts with specific questions to trace incidents back to the process failures that caused them. It then applies a modified version of Events and Causal Factors (ECF) analysis from the Department of Energy [23],

which is an event based modeling technique. Together, STAMP and PARCEL can be used to identify both the control failures and development process failures that contribute to incidents or accidents.

2.3 Requirement elicitation, specification, and refinement

Requirements engineering is defined by Nuseibeh and Easterbrook as the process of identifying and documenting the purpose of a system "in a form that is amenable to analysis, communication, and subsequent implementation"[67]. The primary tasks of requirements engineering are eliciting, modeling and analyzing, communicating, agreeing, and evolving. Although most of these tasks occur early in the design process (e.g., during the first stage of the V-model shown in Figure 2.1), requirements are often updated as a result of subsequent development activities.

In the emerging field of computer systems engineering, Ross and Schoman first identified the need for a requirements development practice more similar in rigor to the one applied in manufacturing processes [75] [43]. They called for requirements definition to include context analysis (the objective of the system), a functional specification (what the system should do), and design constraints (how the system should be built), with full documentation of each component. Ross introduced the Structured Analysis (SA) language as a framework for modeling understanding, not only for requirements development but also for any other work that produces an intellectual product [74]. SA uses both text and graphical representations of software components and functions. It also allowed for hierarchical representation of system designs.

Feather introduced formal reasoning about specification of agents in composite systems [26]. He proposed separate specifications for describing the decomposition of a system into components and for describing composite system behavior. Specifications of

component behaviors were derived by pruning (eliminating behaviors which violate some constraint) and decomposing constraints to assign to agents.

2.3.1 Specifying non-functional requirements

Mylopoulos et al. proposed a framework for specifying non-functional requirements in composite systems [65]. In their process-oriented approach, non-functional requirements were defined in terms of goals, links between goals, methods for goal refinement, correlation rules, and a labeling structure for linking goals to design decisions. It also included a detailed decomposition of two types of non-functional requirement, accuracy goals (information correctness) and performance goals (execution time and storage space). Similar to ICPA, accuracy goals were decomposed by assigning a separate goal to each component of the information (e.g. report accuracy was divided by report types, report sections, and function and input parameters). Performance decomposition divides the performance goal as a sum-of-parts (e.g., response time goals for an operation are the sum of the response times of the different parts of the operation).

Leveson's intent specifications are another approach to specifying non-functional requirements [51]. Intent specifications provide hierarchical system abstraction that relies on means-to-ends representation, rather than part-to-whole. Various levels in the intent hierarchy include system purpose, system design principles, black box behavior, physical and logical function, and physical realization. Safety-related constraints are defined as system design principles. This approach provides a framework for expressing system safety goals, but does not offer tactics for safety goal decomposition.

Bitsch attempted to make formal specification of safety requirements easier for non-formalists by introducing a set of formal temporal logic patterns that map to safety requirements written in a natural language [9][10]. Table 2.1 displays the classification scheme.

Table 2.1. Safety requirements pattern classification scheme from [10]

<table>
<tr><td colspan="4">Static Safety Requirements (Invariants)</td></tr>
<tr><td rowspan="5">Dynamic Safety Requirements</td><td rowspan="4">Safety Requirements with Temporal Dependencies</td><td rowspan="3">Safety Requirements about Chronological Succession</td><td>Beginning of Validity</td></tr>
<tr><td>Duration of Validity</td></tr>
<tr><td>Beginning & Duration</td></tr>
<tr><td rowspan="2">Safety Requirements with Explicit Time</td><td>Time Triggering</td></tr>
<tr><td>Event Triggering</td></tr>
<tr><td colspan="3">Safety Requirements about General Access Guarantee</td></tr>
</table>

A disadvantage of formal specification is that non-formalists may be reluctant to use formal requirements, particularly in the early analysis and design stages when creating safety requirements is begun [10][31]. Formalizing an existing requirement written in natural language may be easier than trying to identify new requirements entirely within a formalized framework.

In Bitsch's technique, requirements are first classified as static or dynamic based on whether they apply to one or multiple system states. Dynamic requirements are further classified based on the type of temporal dependency. Each classification is mapped to a specific formula in Computation Tree Logic (CTL). The catalog of patterns provides a clear explanation of the logical expression in natural language. Although the primary purpose of the technique is to translate from natural language to temporal logic, the patterns could be applied in reverse to invariants obtained from pattern reduction.

2.3.2 Goal Oriented Requirements Engineering (GORE)

This thesis builds primarily upon Goal-Oriented Requirements Engineering (GORE), introduced by Dardenne, van Lamsweerde, and Fickas [18][44]. In this approach, requirements engineers first define a set of high-level goals (objectives to be achieved by the composite system). Agents are entities within the system or external users of the system that perform the actions needed to achieve the goals [18][46][47]. At the highest level, these objectives

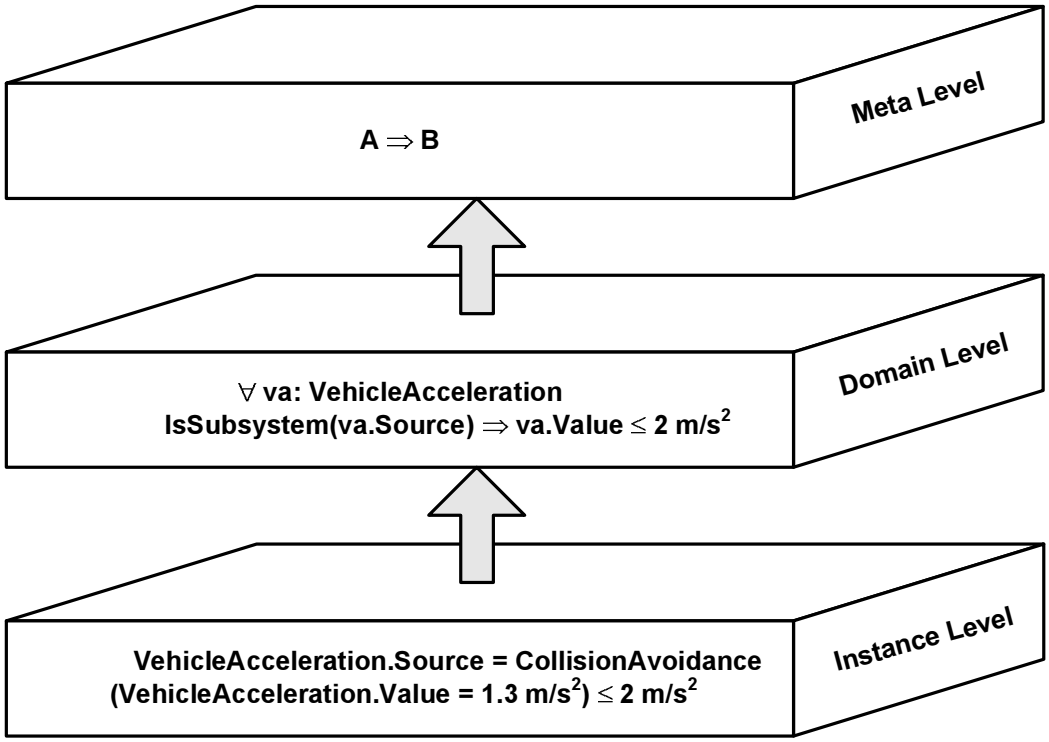


Figure 2.4. Modeling levels for a goal in GORE/KAOS

are nonoperational; no single agent within the system can achieve the goal alone. Through requirements elaboration, goals and subgoals are operationalized into actions (the behaviors that achieve the goal) and refined until they can be assigned to individual agents (a subsystem, a software component, the user, etc.) [19][20][45][46][47][48][49].

GORE operates in three levels of system modeling: the meta, domain, and instance levels [18]. These levels are shown in Figure 2.4 for a single goal. The highest level, the meta level, refers to the abstract concepts such as "goals," "agents," and "actions," and how they interact. The domain level refers to a specific instance of the meta-model, such as a set of specific goals and agents for a given system implementation. Finally, the instance level refers to a specific instance of the model defined at domain level. This could be an instance of an action performed by an agent at specific time during system execution.

P	true in current state
¬P	false in current state
●P	true in previous state
♦P	true in some previous state
■P	true in all previous states
❍P	true in next state
◊P	true in current or some future state
❑P	true in current and all future states
P ∧ Q	P *AND* Q
P ∨ Q	P *OR* Q
P → Q	P implies Q in current state
P ⇒ Q	❑(P → Q); P implies Q in all states
P ⇔ Q	P *iff* Q in all states
●■$_{<T}$P	true for duration T in previous state
●♦$_{<T}$P	true at least once in duration T in previous state
@P	●¬P ∧ P; true in current state, false in previous state
S_0 ⊨ P	True in the initial state
Γ ⊢ P	P is syntactically derivable from the premises Γ
Γ ⊬ P	P is not syntactically derivable from the premises Γ

Figure 2.5. Temporal logic operators

Knowledge Acquisition in autOmated Specification (KAOS) framework. KAOS is the requirements specification language used in GORE [18]. KAOS includes formal definitions in temporal first-order logic [42] for components of the system meta-model, including agents, actions, entities, and goals, as well as the relationships between them. Figure 2.5 lists the temporal operators of KAOS. Formal representation makes system goals explicit and facilitates analysis. An example of a goal for a train control system from [44] is shown in Figure 2.6.

Goal: Achieve[TrainProgress]
InformalDef: *The train shall progress through consecutive blocks.*
FormalDef: (∀ tr: Train, b: Block) [On(tr, b) ⇒ ◊On(tr, b+1)]

Figure 2.6. Goal Achieve[TrainProgress] from [44]

Table 2.2. Goal pattern classifications from [20]

Goal Class	GoalPattern
Achieve	$P \Rightarrow \Diamond Q$
Cease	$P \Rightarrow \Diamond \neg Q$
Maintain	$P \Rightarrow \Box Q$
Avoid	$P \Rightarrow \Box \neg Q$

Refinement patterns. One advantage of using formal specifications is that goal structure can guide elaboration. Darimont and van Lamsweerde identified formal refinement patterns for elaborating goal-oriented requirements [19][20]. They identified four general patterns for goals: *achieve, cease, maintain,* and *avoid*, as shown in shown in Table 2.2 [19, 20].

In this context, safety goals are expressed in the *avoid* pattern ($P \Rightarrow \Box \neg Q$), where P represents the predicate, Q represents the hazardous condition to be avoided, $\Box$ signifies something that will hold always in the future, $\Rightarrow$ signifies *entails*, and $\neg$ is the *NOT* operator.

High-level goals are refined into sub-goals by *AND/OR* graph reduction from the field of Artificial Intelligence [66], by applying formal refinement patterns for logical expressions [19, 20], and by identifying subgoals that are realizable by particular agents [47]. In *AND/OR* reduction, goals are broken-down into a set of subgoals that all must be satisfied (connected with an *AND* operator) or a set of alternative goals (connected with an *OR* operator). Formal refinement patterns identify possible combinations of subgoal patterns that meet the parent goal. For example, a parent goal in the *achieve* form ($P \Rightarrow \Diamond Q$), where $\Rightarrow$ indicates a material implication and $\Diamond$ signifies *eventually*, could be replaced by two *achieve* subgoals of the forms ($P \Rightarrow \Diamond R$) and ($R \Rightarrow \Diamond Q$).

Darimont and van Lamsweerde offer a refinement pattern specifically for safety goals in which safety goals are refined by introducing subgoals for monitoring and responding to an alarm event [20]. The alarm is triggered early enough to give the response mechanism a finite period of time to correct the problem and prevent the original safety goal from

being violated. This provides a template for a general monitoring structure, but gives little guidance on how to define what to monitor. In addition, it allocates all responsibility for achieving the goal to the alarm/response agents. Although this is a common technique for hazard reduction, it is not the only one and may not be desirable for all systems.

Goal realizability. Not every set of subgoals that can be defined to satisfy a parent goal is actually realizable in the system. In the KAOS framework, goal realizability is driven by monitorability and controllability of system state variables. Letier and van Lamsweerde defined a goal to be strictly realizable by an agent if the agent has the ability to monitor all state variables to be monitored and control all state variables to be controlled that are necessary to satisfy the goal [46][47]. More formally, a goal relation can be expressed as $G_{(M,C)}$, where G is the goal to be realized in the system, M is the set of variables in the goal to be monitored, and C is the set of variables in the goal to be controlled. A goal is realizable by an agent if M is a subset of the variables monitored by agent ag and C is a subset of the variables controlled by agent ag, ($M \subseteq Mon(ag)$ and $C \subseteq Ctrl(ag)$).

Letier and van Lamsweerde classified unrealizable goals into the following categories: *lack of monitorability*, *lack of control*, *reference to future*, *goal unsatisfiability*, and *not finitely violable goal*. Consider a goal of the form ($A \Rightarrow B$). Realizability of this goal requires an agent with the ability to control both A and B. If an agent has the ability to monitor A and control B, the goal is still not realizable due to a reference to the future (the agent cannot monitor A and control B in the same state). The goal $\bullet A \Rightarrow B$ could be realizable by an agent with the ability to monitor A and control B. Letier and van Lamsweerde also provide several goal patterns to help identify and resolve goal unrealizability.

2.4 Emergence

Safety goal elaboration is complicated by the emergent nature of system safety. Emergence had been observed in natural systems as early as the time of Aristotle, who recognized there were systems in which, "the whole is something beside the parts" [5]. The term *emergent* was first defined by Lewes [57], who asserted that a *resultant* system could be expressed as a sum or product of its components' outputs, whereas the outputs of components in an *emergent* system are fundamentally different, both from each other and from the resulting system behavior:

> Add heat to heat and there is a measurable resultant; but add heat to different substances, and you get various effects, qualitatively unlike...

Emergence has also been studied in complex systems such as artificial life [16][21] and agent-based systems [72]. Cariani [16] differentiates among three types of emergence: *computational*, *thermodynamic*, or *relative to a model*. Whereas computational emergence focuses on identifying emergent behaviors that arise from local computations, emergence relative to a model is more concerned with how emergent properties of the system affect the behavior of its components.

Darley [21] asserts that, similar to the halting problem [82], determining whether or not a system is emergent is undecidable, and the best predictor of system behavior is simulation. It may be possible to observe emergent behavior (something that is not predicted by individual component behaviors) as the system runs, but it is impossible to know for sure that a system is non-emergent, or even that all emergent behaviors in the system have been exposed.

Privosnik [72] proposed defining emergence at various levels of the hierarchy in agent-based systems. Component behaviors contribute to emergent behaviors at the subsystem

level, which contribute to new emergent behaviors at the system level. Thus, system behavior is achieved by defining the behaviors of combinations of agents, as well as the agents themselves.

2.4.1 Feature interaction

Bowen et al. identified emergent behaviors that resulted from the integration of new features into existing telecommunication systems as *feature interactions* [13]. A feature is "a package of functionality incrementally added to a service to enhance or modify it" [84]. In a telecommunication system, caller-ID and call-waiting are examples of features that have been added to basic telephone service (also known as Plain Old Telephone Service or POTS). New interactions arise when features share resources, such as signaling inputs or network messages, or when new behaviors violate feature assumptions [15]. This feature interaction problem applies to automotive systems as well. For example, new features that introduce autonomous vehicle control, such as Adaptive Cruise Control (ACC) or Collision Avoidance (CA) may violate previous assumptions that all brake commands are driver-initiated. Likewise, multiple features may have conflicting throttle commands.

Zave suggested the use of formal specifications to eliminate many feature interactions in telecommunication systems [88]. The basic idea is to eliminate ambiguities in feature behavior or assumptions by formalizing them. In the approach, system-level invariants are proposed as a means of limiting unwanted behavior across features. Jackson and Zave proposed an architecture for composing features to eliminate unwanted feature interactions called Distributed Feature Composition (DFC) [36]. In this architecture, features are modeled as components with inputs and outputs. Calls between customers are directed by a central router from feature to feature until completion. Feature interactions are limited by the order in which the call was routed through the features. Although this may work

in telecommunication systems where each call has a source and destination, it is unclear whether this pipe-and-filter architecture would work for automotive systems with closed-loop control.

2.5 Safety goal monitoring

Feather et al. proposed a run-time monitoring system based on goal-driven requirements [27]. In their approach, monitored parameters that can be violated are identified at design time, and alternative system designs for recovery actions are explored. At run time, the violated properties are reconciled by choosing an alternative requirement to enforce, or by changing control parameters to restore the original invariant. Although the general process is outlined, the approach does not identify specific tactics for choosing invariants to monitor. Also, the run-time system adaptation seems better suited for systems that are not safety critical.

Plale-Schroeder et al. proposed using a database query language to check safety constraints at run-time [78][70][69]. Safety constraints derived from hazard analysis are defined as a triggering event, a set of conditions for a response, and a response action. The technique was applied to a set of autonomous robots in which monitoring of constraints was centralized. Although a database query language seemed well-structured for checking constraints and performing recovery actions, it would not be easy to use in many embedded systems where there are no existing similar systems and computing resources are limited.

Kim et al. proposed a run-time monitoring technique, Monitoring and Checking (MaC), based on formal system models [40]. The basic idea is to use a model-checker on real system values at run-time, rather than on a model of the specification, to verify that the software is meeting its formal requirements. This required mapping of the low-level system state variables to model state. Monitor scripts are written in Primitive Event Definition

Language (PEDL) and contain a monitored entity (software method, class, etc.), conditions (statements that evaluate to true or false), and events (generated when state variables change). Sokolsky et al. extended this work by defining index and attribute quantifiers for each event [79]. Index quantifiers bind groups of events (e.g. airbag deployed messages from a particular vehicle) and attribute quantifiers bind a variable in the checked formula to the most recent occurrence of the event (e.g. the last airbag deployed message from a particular vehicle). Whereas these techniques require a formal specification of the system design, my goal is to provide run-time monitoring that will work whether or not a formal design specification exists. Also, the focus of my approach is not functional correctness, but rather the much narrower property of conformance to safety requirements.

Peters and Parnas [68] identified issues with requirements-based monitors for real-time systems. Because physical devices have some range of precision, monitors that observe those devices may have a range of outputs depending on whether they use the best-case value, worst-case value, or something in between. This could also affect monitor accuracy, as pessimistic monitors may be more likely to trigger false positives (indicate the system is not functioning according to the requirements when it is). Likewise, optimistic monitors may be more likely to trigger false negatives (indicate that the system is functioning according to the requirements when it is not). Another problem arises when timing differences between the monitor and the system lead to nondeterminism, if the monitor samples environmental variables independently of the target software system. This is not an issue for monitors that share inputs with the software system being observed. Although the monitors described by Peters and Parnas were non-hierarchical, the issues they identified are relevant to my approach.

2.5.1 Safety kernels

Leveson et al. introduced the idea of a safety kernel to detect and recover from safety-critical errors [56]. In their approach, assertions to check safety invariants are embedded in the application software. A separate timer is used to monitor real-time requirements. When a hazard is detected, the safety kernel executes a recovery action. One of the main purposes of the safety kernel is to isolate the recovery mechanisms and the policies governing them from the rest of the system. This reduces complexity for analysis and testing of critical functions. My approach differs in that the monitoring functions are separate from the subsystems being monitored, rather than embedded in the application code as assertions. Also, there is no centralized recovery mechanism; both monitoring and response in my approach are distributed across the subsystems. Their version of the safety kernel design was neither implemented nor evaluated in a real safety-critical system.

Rushby proposed using safety kernels to enforce, rather than monitor, safety properties of the system [76]. He proposed that a safety kernel could be used to prevent negative behaviors from happening if all actuator commands were required to pass through the safety kernel. This type of system could not be used to to ensure positive behaviors of the system. The approach provided a general description of a safety kernel structure, along with some suggestions for enforcing event sequencing, but the ideas were not evaluated in a real system implementation. Wika and Knight extended the ideas proposed by Rushby by classifying safety policies by fault type [86, 87]. They also provided a mechanism for weakened policies whereby some of the enforcement is transferred to the application. The work included application of the approach to two case studies, a neurosurgical device and a nuclear research reactor, and a dependability evaluation of the safety kernel. As with Rushby's design, the safety kernel requires exclusive control over application devices. As a single point of failure, this requires the safety kernel to exhibit a high degree of depend-

ability.

The safety kernel approach differs from my work in a number of ways. First, the focus of the safety kernel is enforcement of safety policies, not monitoring. A true enforcement kernel requires exclusive control of system actuation, something that is not practical in many distributed embedded systems. The automotive application domain, with its distributed design and development processes is not likely to transition to an architecture that will permit this type of centralized control. A second difference is the fail-stop nature of the safety kernel; safety policies are enforced by blocking unsafe commands to actuators. Once again, this may not work in some application domains where complete loss of the application may be worse than a potential hazard caused by an occasional incorrect command. Third, the safety kernel approach is centralized by principle, whereas my run-time monitors are distributed across multiple subsystems. Finally, the approaches to safety kernels leave identification of the monitored properties to future work. As part of my results, I provide specific techniques for identifying the safety invariants to be monitored.

Chapter 3

Emergence in Composite Systems

3.1 Overview

In general, emergence is problematic because the behaviors it produces are unexpected. In telecommunications, emergence has been identified as the feature interaction problem, when adding features to a system produces new behaviors not included in any single feature, or where the behavior of one feature is changed by the behavior of another [13]. For example, features that hide call origination information may interfere with an automatic call-back feature [15].

In safety-critical systems, emergence causes two types of problems. The first occurs when interaction among components produces hazardous behaviors that are previously unidentified. In this situation, one or more system safety goals are missing, which means the set of subgoals obtained by goal elaboration will also be incomplete. The second problem occurs when interaction among components produces emergent behaviors that violate a known system safety goal (i.e., the system was designed and built to satisfy the goal but ultimately did not). In this case, the parent goal is not fully satisfied by the set of subgoals for the components and their relationships to each other. This thesis addresses the second problem: emergence that prevents a known system safety goal from being satisfied.

3.1.1 Motivation

In order to understand and handle emergence in system safety, it is useful to define emergent and composable system behaviors within the context of the formal specification of the safety goals because this makes explicit:

- uncertainty due to emergence from unknown dependencies or unrealizable goals
- goal redundancy to mitigate unknown emergence
- restriction tactics for handling unrealizable emergence

The aim of requirements elaboration is to precisely define the behavior of the system under development. When formal definitions are used in the requirements specification, these requirements can be mathematically verified. However, in any requirements specification there is some degree of uncertainty. This uncertainty arises when assumptions supporting the elaboration process fail, or when there are unanticipated dependencies between goals. Formal specification of emergence acknowledges the presence of this uncertainty in the system.

In goal elaboration, particularly for system safety goals, it may be desirable for certain behaviors to be redundantly met by the system design. Redundancy may come in the form of redundant functional behaviors, or redundant constraints on different functional behaviors. Formal specification of emergence provides a platform for including goal redundancy in the system design.

Some known subgoals for a decomposition of a parent goal may be unrealizable in the system. To guarantee a goal is met, alternate restrictive goals may be chosen to remove the emergence from the functional design. Formal specification of emergent behaviors assists in identifying these situations and alternative restrictive subgoals.

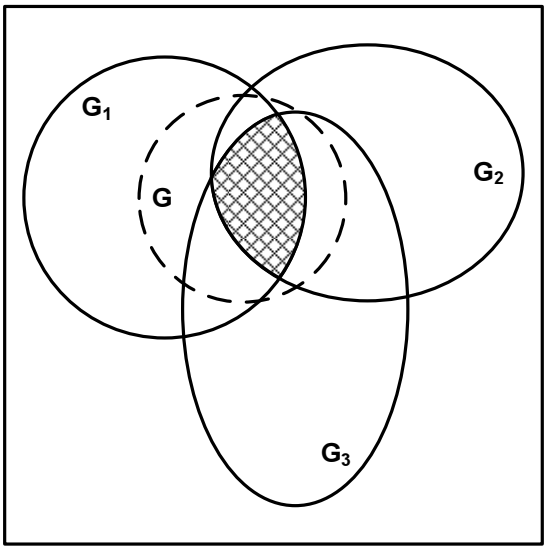


Figure 3.1. Complete and-reduction $\{G_1, G_2, G_3\}$ of goal G

This chapter addresses the following questions posed in Chapter 1:

- What does it mean for a goal to be composable or emergent?
- When is it possible to decompose a system-level goal?
- Is partial decomposition possible, and if so, is it useful?

3.1.2 And-reductions

Darimont formally specified **and-reductions** as a representation of decomposition for pattern-based goal refinement [19]. In his definition of decomposition, goals $\{G_1, G_2, ...G_n\}$ are a **complete and-reduction** of goal G iff:

1. $G_1, G_2, ...G_n \vdash G$

   The parent goal can be inferred from the conjunction of the subgoals.

2. $(\forall i, j)[\wedge_{j \neq i} G_j \nvdash G]$

   The subgoals are minimally sufficient for inferring the parent goal.

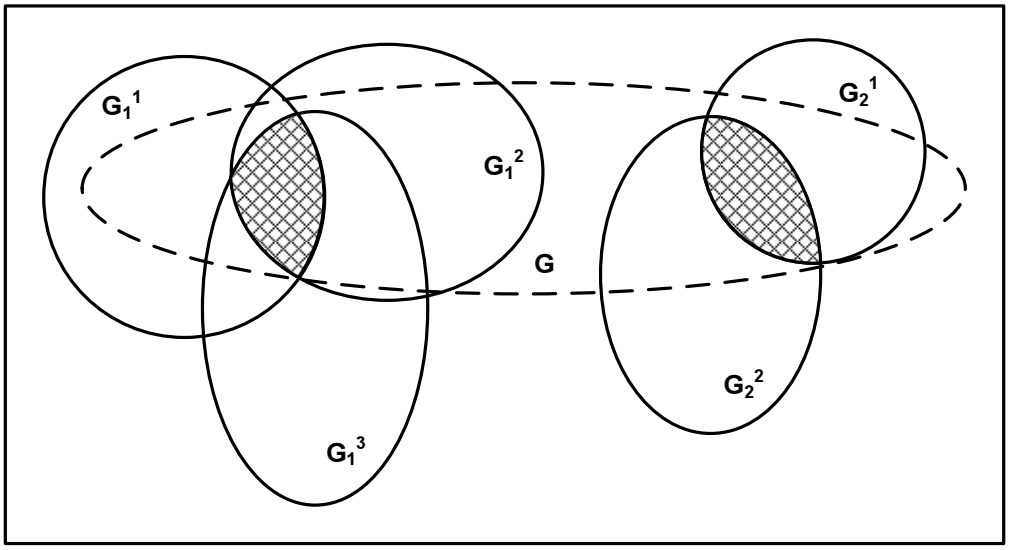


Figure 3.2. And-reductions $\{G_1^1, G_1^2, G_1^3\}$ **and** $\{G_2^1, G_2^2\}$ **of goal** G

3. $G_1, G_2, ...G_n \nvdash false$

   The subgoals are not mutually incompatible.

4. $if\ (G \Rightarrow \wedge_{1 \leq j \leq n} G_j)$ then either $n > 1$ or the proof relies on some domain knowledge

   Simple restatement of the parent goal is not a decomposition

Figure 3.1 shows the state space of a complete and-reduction of goal G consisting of subgoals $\{G_1, G_2, G_3\}$. Because one particular and-reduction is minimally sufficient, but not necessary for satisfying the parent goal, there may be multiple and-reductions for the same goal (i.e., other sets of subgoals that also satisfy the parent goal). Figure 3.2 shows the state space of two complete and-reductions of goal G consisting of subgoals $\{G_1^1, G_1^2, G_1^3\}$ and subgoals $\{G_2^1, G_2^2\}$.

The and-reductions are mapped over different state variables. Suppose goal G in Figure 3.2 is $(A \Rightarrow B)$. Table 3.1 shows possible values for subgoals $G_1^1, G_1^2, G_1^3, G_2^1$, and G_2^2. Note that $\{G_1^1, G_1^2, G_1^3\}$ maps over state space defined by the state variables $\{A, B, C, D\}$, and $\{G_2^1, G_2^2\}$ maps over state space defined by the state variables $\{A, B, E\}$.

Table 3.1. Subgoals $G_1^1, G_1^2, G_1^3, G_2^1, G_2^2$ for goal G

Goal	Subgoals		
G: A ⇒ B	G_1^1: A ⇒ C	G_1^2: C ⇒ D	G_1^3: D ⇒ B
G: A ⇒ B	G_2^1: A ⇒ E	G_2^2: E ⇒ B	

Darimont further defined a **partial and-reduction** of goal G as the set of subgoals $G_1, G_2, ...G_m$ iff:

1. $\exists G_{m+1}, ...G_n$ such that $\{G_1, ...G_m, G_{m+1}, ..., G_n\}$ is a complete and-reduction

Given (2) in the conditions for a complete and-reduction, the parent goal cannot be inferred from a partial and-reduction alone. In Figure 3.1, if subgoal G_2 is missing, G cannot be inferred.

3.1.3 Scope

In this chapter, goals are defined as propositional logic expressions of system state, using the operators listed in Figure 2.5. Within the KAOS framework of goal-oriented specification, these definitions appear at the domain level (i.e., for a specific decomposition of the parent goal in the application domain). The system decomposition should reflect all behaviors the system produces and should prohibit any additional behaviors that are unspecified.

This chapter presents formal definitions of goals that are:

1. fully composable
2. fully composable with redundancy
3. emergent
4. emergent but partially composable

5. emergent but partially composable with redundancy

3.2 Composable goals

At the meta level of the KAOS framework [18], an and-reduction represents one decomposition out of several possible decompositions of the parent goal. In pattern-based goal decomposition, and-reduction patterns can be applied to generic goal patterns to identify missing subgoal patterns. However, at the domain level of the KAOS framework [18], an and-reduction representation is an inexact decomposition of the parent goal because the decomposition it defines may not be realizable in the system, may not be chosen for a given system design, or may be one of multiple redundant decompositions chosen by the requirements engineers.

An exact decomposition of a goal in a requirements specification not only defines when the goal is satisfied, but also when the goal is not satisfied. In other words, it defines both the behavior of G and the behavior of $\neg G$. In and-reduction, if one subgoal is not satisfied (i.e., evaluates to $FALSE$), then the remaining subgoals are not sufficient conditions to allow one to infer the parent goal. It is undefined whether or not the parent goal is satisfied, because there may be other and-reductions that do satisfy the parent goal. As the design becomes more complete, the number of possible alternative decompositions decreases. An exact decomposition of a goal defines which particular decomposition, restrictive decomposition, or combination of redundant decompositions is chosen by the requirements engineers for the system design.

In some situations, inexact decomposition may not be problematic. In hazard detection, for example, if the subgoals are unable to prove the system is in a safe state, a conservative design approach would be to assume the system is in an unsafe state and commence hazard mitigation and recovery actions. In essence, for hazard detection it is assumed that the

and-reduction subgoals are not only minimally sufficient, but also necessary conditions for satisfying the parent goal.

For other purposes, however, simply assuming the behavior is not occurring may be a problem. For example, suppose one system goal requires a brake to be applied when an object is detected in the vehicle path, and that an and-reduction is chosen that assigns that functionality to a collision avoidance subsystem. Another subsystem may have the capability to achieve the same goal (e.g., an adaptive cruise control system with object detection and braking abilities), but it is not desirable to have that subsystem do so without making it an explicit goal for that subsystem. This is problematic for two reasons. First, analysis of the behavior would be incomplete if other components produced the behavior in a manner that is unspecified. Second, other goals and subgoals dependent on the behavior could be incorrect or incomplete. For a system requirements specification, the definition of decomposition should be more precise.

3.2.1 Fully composable

This thesis defines goal G to be **fully composable** if there exists a set of n subgoals $\{G_1, G_2, G_3, ...G_n\}$ that are realizable by one or more components, such that G is materially equivalent to the conjunction of the subgoals:

$$G_1 \wedge G_2 ... \wedge G_n \Leftrightarrow G \tag{3.1}$$

which is equivalent to the conjunction of expressions:

$$G_1 \wedge G_2 ... \wedge G_n \Rightarrow G \tag{3.2}$$

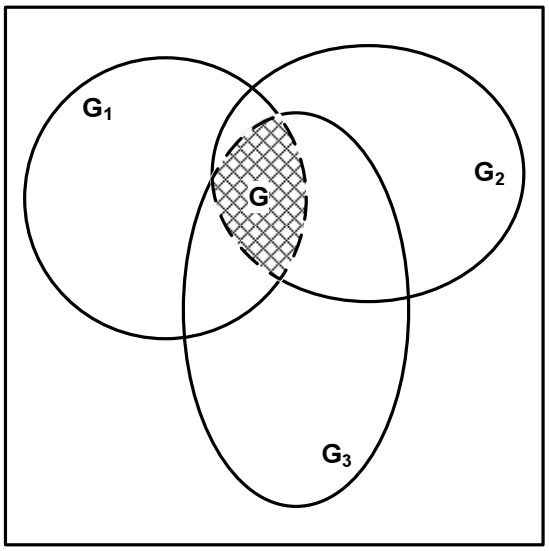


Figure 3.3. Goal G, fully composable by goals $\{G_1, G_2, G_3\}$

and

$$\neg G_1 \vee \neg G_2 ... \vee \neg G_n \Rightarrow \neg G \tag{3.3}$$

Thus, the parent goal is satisfied (evaluates to *TRUE*) when all subgoals are satisfied and is violated (evaluates to *FALSE*) when any subgoal is violated. This means that the parent goal is satisfied by exactly one and-reduction, and that other and-reductions are prohibited in the system specification. A goal G that is fully composable by goals $\{G_1, G_2, G_3\}$ is illustrated in Figure 3.3.

Consider the goal that requires a brake to be applied when an object is in the vehicle path:

$$ObjectInPath \Rightarrow StopVehicle \tag{3.4}$$

Subgoals that fully compose the goal for a collision avoidance subsystem are:

$$ObjectInPath \Leftrightarrow CA.StopVehicle \tag{3.5}$$

and

$$CA.StopVehicle \Rightarrow StopVehicle \tag{3.6}$$

These subgoals state that the collision avoidance always stops the vehicle when an object is in the vehicle path, and only when an object is in the vehicle path.

3.2.2 Fully composable with redundancy

If redundant behaviors are required, that is if multiple and-reductions are chosen to satisfy the parent behavior, a new definition for decomposition is required. Let G, G_i, and G_i^j be goals such that:

$$G_i \in \{G_1, G_2, ...G_p\} \tag{3.7}$$

and

$$G_i \Leftrightarrow (G_i^1 \wedge G_i^2 ... \wedge G_i^q) \tag{3.8}$$

This thesis defines goal G to be **fully composable with redundancy** iff:

$$G_1 \vee G_2 ... \vee G_p \Leftrightarrow G \tag{3.9}$$

which is equivalent to the conjunction of expressions:

$$G_1 \vee G_2 ... \vee G_p \Rightarrow G \tag{3.10}$$

and

$$\neg G_1 \wedge \neg G_2 ... \wedge \neg G_p \Rightarrow \neg G \tag{3.11}$$

If one or more and-reduction G_i is satisfied, then the parent goal will also be satisfied.

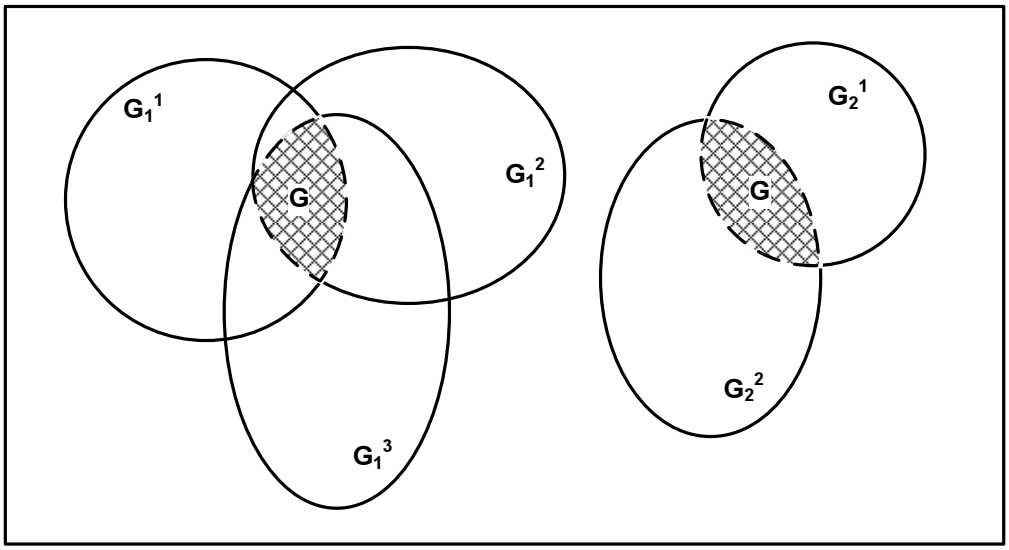


Figure 3.4. Goal G, fully composable with redundancy by goals $\{G_1^1, G_1^2, G_1^3\}$ and $\{G_2^4, G_2^5\}$

Likewise, if none of the and-reductions are satisfied, then the parent goal will not be satisfied. All other possible and-reductions are prohibited in the system specification. A goal G that is fully composable with redundancy by and-reductions $\{G_1^1, G_1^2, G_1^3\}$ and $\{G_2^4, G_2^5\}$ is illustrated in Figure 3.4.

Again, consider the goal that requires a brake to be applied when an object is in the vehicle path in Equation (3.4). The goal can be redundantly satisfied by both collision avoidance and adaptive cruise control. The subgoals that fully compose the goal with redundancy are:

$$ObjectInPath \Leftrightarrow CA.StopVehicle \vee ACC.StopVehicle \tag{3.12}$$

and

$$CA.StopVehicle \vee ACC.StopVehicle \Rightarrow StopVehicle \tag{3.13}$$

These subgoals state that the collision avoidance and adaptive cruise control attempt to stop the vehicle when an object is in the vehicle path, either of those attempts will cause the vehicle to stop, and only collision avoidance or adaptive cruise control performs this action.

3.3 Emergent goals

In an ideal world, all system-level goals could be fully composed, with or without redundancy, by known and realizable subgoals. However, in reality systems are never fully composed because some degree of uncertainty will remain in decomposition of any but the simplest systems. For safety-critical systems that interface with or control other highly complex systems, such as humans or the natural world, it may be impossible to remove all emergence because doing so requires definitions of all possible state variables and their relationships to the state variables in the safety goal. Emergence is a consequence of this uncertainty.

We define goal G to be *emergent* if there is no set of subgoals realizable by any component in the system that satisfy Equation (3.1).

3.3.1 Emergent but partially composable

If a safety goal is not fully composable, it may still be possible to isolate some portion of the goal behavior that is partially composable. As discussed in Section 3.1.2, Darimont defines a partial and-reduction as a partial decomposition [19]. In this thesis, partially composable goals are similar to partial-and-reduction.

This thesis defines goal G to be **emergent but partially composable** if there exists a set of m subgoals $\{G_1, G_2, ...G_m\}$ that are realizable by one or more components in the system,

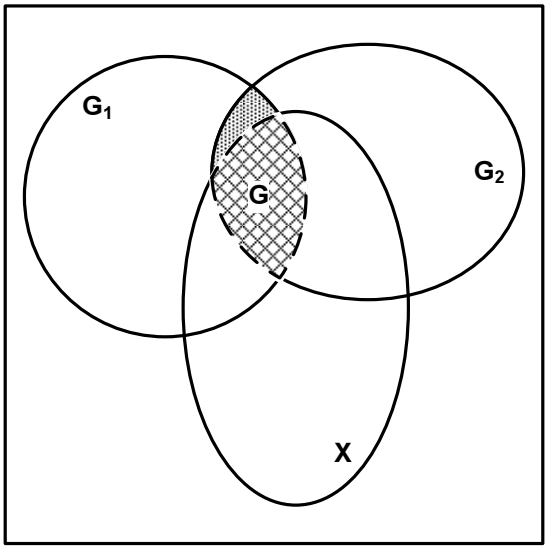


Figure 3.5. Goal G, partially composable by goals $\{G_1, G_2\}$ with emergent behavior X

plus some undefined or unrealizable goal(s) X, such that G is materially equivalent to the conjunction of the realizable and undefined subgoals:

$$G_1 \wedge G_2 ... \wedge G_m \wedge X \Leftrightarrow G \tag{3.14}$$

which is equivalent to the conjunction of expressions:

$$G_1 \wedge G_2 ... \wedge G_m \wedge X \Rightarrow G \tag{3.15}$$

and

$$\neg G_1 \vee \neg G_2 ... \vee \neg G_m \vee \neg X \Rightarrow \neg G \tag{3.16}$$

In this definition, emergence exists as hidden or missing subgoals within the and-reduction. A goal G that is partially composable by goals $\{G_1, G_2\}$ with emergent behavior X is illustrated in Figure 3.5. If X evaluates to *FALSE* (i.e., there exists some unknown dependency

Table 3.2. Subgoals $G_1^1, G_1^2, G_1^3, G_2^1, G_2^2$ for goal G, with emergence X_1 and X_2

Goal	Subgoals			
G: A $\Rightarrow$ B	G_1^1: A $\Rightarrow$ C	G_1^2: C $\Rightarrow$ D	G_1^3: D $\Rightarrow$ B	X_1
G: A $\Rightarrow$ B	G_2^1: A $\Rightarrow$ E	G_2^2: E $\Rightarrow$ B	X_2	

between the known subgoals and some unknown subgoal, and that unknown subgoal does not evaluate to *TRUE*), then the parent goal is not satisfied, even if the defined subgoals G_1 and G_2 are satisfied.

For emergent but partially composable goals, emergence works against goal satisfaction. That is, some emergent behavior is required, in addition to satisfying the known subgoals, in order to satisfy the parent goal. Darimont's reduction patterns [19] can be used to identify missing subgoals in a partial and-reduction, based on the patterns of the previously identified subgoals. However, even if a goal G seems to be a complete and-reduction that is realizable in the system, there remains some probability that one or more of the subgoals are not realizable for all possible system states due to unknown dependencies.

For example, consider the subgoals listed in Table 3.1. Table 3.2 shows the same subgoals with emergence acknowledged. Suppose state variable C is dependent on some other state variable F, defined by the expression $(F \Rightarrow \neg C)$. If this dependency is unknown at the time of goal elaboration, the partial and-reduction $\{G_1^1, G_1^2, G_1^3\}$ may appear to be a complete and-reduction. However, the dependency between F and C becomes an assumption of the decomposition, essentially serving as a subgoal (G_1^4). With this added assumption, a missing subgoal for the decomposition is $\square\neg F$, (G_1^5). In Table 3.2, these two new subgoals exist in X_1.

Consider again the goal defined in Equation (3.4). Uncertainty in object detection can

be expressed as a latent dependency between the object position and collision avoidance:

$$ObjectInPath \Leftrightarrow CA.ObjectInPathDetected \vee CA.ObjectInPathNotDetected \quad (3.17)$$

Now, the actual goals that must be met in the system are:

$$CA.ObjectInPathDetected \Rightarrow CA.StopVehicle \quad (3.18)$$

$$CA.ObjectInPathNotDetected \Rightarrow CA.StopVehicle \quad (3.19)$$

$$CA.StopVehicle \Rightarrow StopVehicle \quad (3.20)$$

where Equation (3.19) exists in X.

In order for a goal to be fully composable, without emergence, all possible dependencies between all possible state variables must be known and explicitly defined.

3.3.2 Emergent but partially composable with redundancy

Emergence in a system not only appears as an inhibitor of goal satisfaction, but also sometimes serves as a redundant source of goal satisfaction. Let G, G_i, and G_i^j be goals, and X_i and Y be some undefined or unrealizable goal(s) such that:

$$G_i \in \{G_1, G_2, ...G_r\} \quad (3.21)$$

and

$$G_i \Leftrightarrow \{G_i^1 \wedge G_i^2 ... \wedge G_i^s \wedge X_i\} \quad (3.22)$$

This thesis defines goal G to be **emergent but partially composable with redundancy**

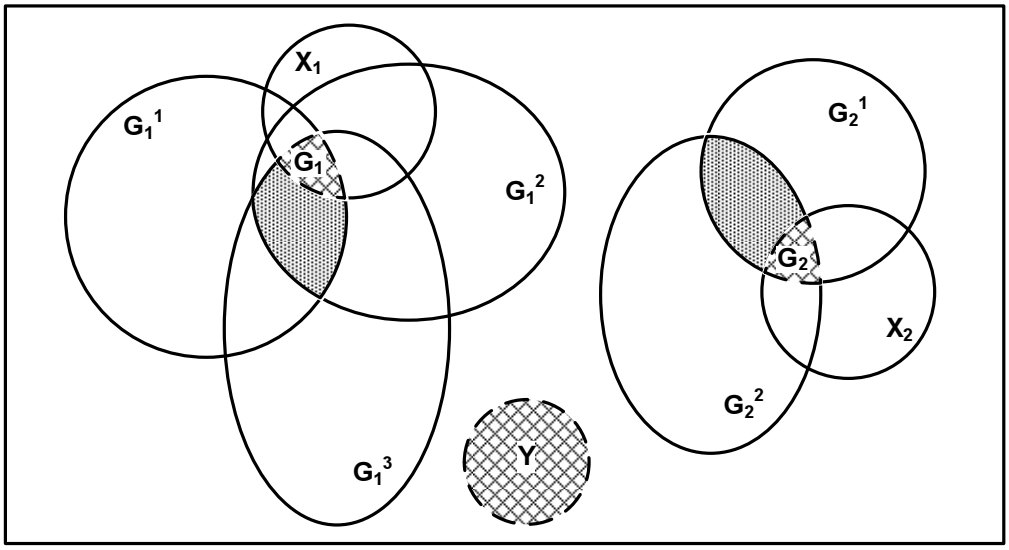


Figure 3.6. Goal G, partially composable with redundancy by goals $\{G_1^1, G_1^2, G_1^3\}$, $\{G_2^1, G_2^2\}$ with emergent behaviors X_1, X_2, and Y

iff:

$$G_1 \vee G_2 ... \vee G_r \vee Y \Leftrightarrow G \tag{3.23}$$

which is equivalent to the conjunction of expressions:

$$G_1 \vee G_2 ... \vee G_r \vee Y \Rightarrow G \tag{3.24}$$

and

$$\neg G_1 \wedge \neg G_2 ... \wedge \neg G_r \wedge \neg Y \Rightarrow \neg G \tag{3.25}$$

Figure 3.6 illustrates a goal G that is emergent but partially composable with redundancy. Whereas X_i represents hidden dependencies that make the and-reductions of G incomplete, Y represents hidden dependencies and behaviors that satisfy G when no other defined and-reduction of G is satisfied. Intuitively, emergent behavior Y serves as an "angel" in the system that satisfies the parent goal when the system specification fails to do so, and emer-

gent behavior X_i serves as a "demon" that prevents the parent goal from being satisfied, even though all defined system goals evaluate to *TRUE*.

In some cases, Y results when it is too difficult to exclude all unspecified behavior in the system design and implementation. In the example from Section 3.2.1, even if collision avoidance is the only subsystem assigned to satisfy the goal, adaptive cruise control may happen to stop the vehicle at the same time an object is in the vehicle path. It may be possible to prevent adaptive cruise control from accidentally satisfying the goal, perhaps by prohibiting adaptive cruise control from performing any action when an object is in the vehicle path, but it may not always be practical to do so.

Consider again the goal defined in Equation (3.4). Taking into account uncertainty in object detection of both collision avoidance and adaptive cruise control (X_1 and X_2), and uncertainty of other vehicle behaviors coincidentally causing the vehicle to stop when an object is in the vehicle path, the new assumption is:

$$\begin{aligned} &ObjectInPath \Leftrightarrow \\ &\quad (CA.ObjectInPathDetected \vee CA.ObjectInPathNotDetected) \\ &\quad \wedge (ACC.ObjectInPathDetected \vee ACC.ObjectInPathNotDetected) \end{aligned} \tag{3.26}$$

and the new subgoals are:

$$CA.ObjectInPathDetected \Rightarrow CA.StopVehicle \tag{3.27}$$

$$CA.ObjectInPathNotDetected \Rightarrow CA.StopVehicle \tag{3.28}$$

$$ACC.ObjectInPathDetected \Rightarrow ACC.StopVehicle \tag{3.29}$$

$$ACC.ObjectInPathNotDetected \Rightarrow ACC.StopVehicle \tag{3.30}$$

$$CA.StopVehicle \vee ACC.StopVehicle \vee Unknown.StopVehicle \Rightarrow StopVehicle \quad (3.31)$$

where *Unknown.StopVehicle* is Y and Equations (3.28) and (3.30) are X_1 and X_2

3.3.3 Usefulness of partial composability

Subgoals for a partially composable system goal are not very useful when the objective of the goal is to *produce* a behavior. In order to produce behavior G in Equation (3.23), one of the subgoals G_1 through G_r (each is an and-reduction of G) or the unknown goal(s) in Y would have to be satisfied. But to satisfy any G_i, X_i from Equation (3.22) also has to be satisfied. Without knowing X_i it is impossible to say anything about whether or not G_i is satisfied, and without knowing if any G_i is satisfied, it is impossible to say anything about whether or not G is satisfied.

Alternatively, if the real objective is to *prohibit* a behavior, then a partially composable system goal may be of some use. Safety goals specify the "safe" state of the system to be maintained, but the real objective of system safety is to prevent a hazardous state from occurring. If goal G is some safety goal and any G_i^j evaluates to *FALSE*, then G_i is not satisfied. If there is no goal redundancy (i.e. $r = 1$ in Equation (3.23), or if no G_i is satisfied, then the system may be in a hazardous state. In this situation it is useful to know specific conditions that could lead goal G to be violated, even if it is impossible to know all conditions required to satisfy the goal. Although it is impossible to guarantee the system will always be safe without knowing and guaranteeing at least one G_i (i.e. ensuring all G_i^j *and* X_i are *TRUE* for at least one value of i), it is possible to prevent some known conditions that put the system into a hazardous state.

In other words, it may be possible to specify some, but not all component behaviors or interactions that cause a hazard. For example, it may be known that a vehicle will accelerate

if a subsystem applies the throttle. However, there may be additional unknown interactions in the vehicle that also cause vehicle to accelerate (e.g., loss of brake fluid combined with a slick, sloped road). If a safety goal restricts accelerations over a certain threshold, it is important to prevent known sources of acceleration from violating the goal, in this case by limiting the acceleration caused by a subsystem. Preventing or mitigating one hazard is useful, even if the same cannot be done for others.

3.3.4 Conjunctive goals

In general, if a goal can be expressed as a conjunction of expressions, then it may be partially composable. A parent goal of the form:

$$\square(A \wedge X) \tag{3.32}$$

can be divided into two subgoals:

$$\square A \tag{3.33}$$

and

$$\square X \tag{3.34}$$

Another goal of the form:

$$A \vee X \Rightarrow B \tag{3.35}$$

which is equivalent to:

$$\square((\neg A \wedge \neg X) \vee B) \tag{3.36}$$

can be divided into two subgoals:

$$A \Rightarrow B \tag{3.37}$$

and

$$X \Rightarrow B \tag{3.38}$$

If X is unknown or unrealizable, the behaviors required by goals (3.33) and (3.37) may still be ensured, even if all behaviors required by goals (3.32) and (3.36) cannot. Consider a goal that requires a vehicle to be stopped whenever there is an object in the vehicle path. If object detection is non-ideal, then the goal cannot be fully realized. This goal, including the uncertainty, can be expressed in the same form as Equation (3.35):

$$InPathDetected \ \vee \ InPathNotDetected \Rightarrow \ StopVehicle \tag{3.39}$$

which can be divided into two subgoals:

$$InPathDetected \ \Rightarrow \ StopVehicle \tag{3.40}$$

and

$$InPathNotDetected \ \Rightarrow \ StopVehicle \tag{3.41}$$

In this example, subgoal (3.40) can be realized in the system, even if subgoal (3.41) cannot.

3.3.5 Disjunctive goals

An emergent goal with unknown or unrealizable subgoals may still be satisfied by making it more restrictive in one of two ways. First, an *OR* reduction may be applied to disjunctions [66]. A goal of the form:

$$\Box(A \vee X) \tag{3.42}$$

where X is an unknown or unrealizable subgoal, can be satisfied by the more restrictive goal:

$$\Box A \tag{3.43}$$

Likewise, a goal of the form:

$$A \wedge X \Rightarrow B \tag{3.44}$$

which is equivalent to:

$$\Box(\neg A \vee \neg X \vee B) \tag{3.45}$$

can be satisfied by the more restrictive goal:

$$A \Rightarrow B \tag{3.46}$$

OR reduction is more restrictive because some functionality of the system that would otherwise be acceptable is prohibited. In a system with goal (3.45), the state $A \wedge \neg X \wedge \neg B$ is acceptable, but in a system with goal (3.46) it is not.

Another way to make the goal more restrictive is to increase the safety envelope of a given variable. Consider a goal that prohibits vehicle acceleration above a certain threshold *AccelerationLimit*.

$$\Box(VehicleAcceleration < AccelerationLimit) \tag{3.47}$$

A subgoal for subsystems that control vehicle acceleration might prohibit requests that exceed a lower threshold, perhaps *AccelerationLimit - SafetyEnvelope*.

$$\Box(VehicleAccelerationRequests < (AccelerationLimit - SafetyEnvelope)) \tag{3.48}$$

In this example, some safe acceleration requests are prohibited.

3.4 Composability

In this thesis, **composability** of a goal is defined as the extent to which emergent behaviors, X and Y in Equations (3.14) and (3.23), are small. In all but the most simple systems, some degree of emergence is unavoidable. There will always be a component whose behavior cannot be fully specified or an unanticipated interaction among components. X and Y could exist because you don't know what to look for. Other times, they might occur if observability and controllability requirements of the goal are not met by any subsystem or combination of subsystems smaller than the entire system.

Just as it is impossible to determine whether a system is not emergent, it is also impossible to know if maximum composability has been extracted from an emergent but partially composable goal. In other words, it is impossible to determine if X and Y are minimized. X and Y could be estimated, however by run-time monitoring of the system goal and subsystem subgoals. A violation of a subgoal that does not correspond to a violation of its parent goal is considered a false positive (the subgoals indicate a hazardous state but the system is not in a hazardous state). A violation of a goal that does not correspond to a violation of one or more subgoals is considered a false negative (the subgoals do not indicate a hazardous state but the system is in a hazardous state). False negatives can occur when the unknown/unrealizable subgoal(s) are the cause of the goal violation (X in Equation (3.14)). False positives can occur when the subgoals are more restrictive than the original safety goal (Y in Equation (3.23)).

3.5 Conclusion

This chapter proposed a formal representation of a system decomposition that includes emergence, from both unrealizable and unknown sources, and goal redundancy. This representation included definitions of composable and emergent but partially composable behaviors, with and without redundancy. In addition, the role of emergence in conjunctive and disjunctive goal forms was discussed.

Chapter 4

Indirect Control Path Analysis

4.1 Overview

This chapter presents a new technique for decomposing system safety goals called Indirect Control Path Analysis (ICPA) [11]. ICPA is a top-down search, similar to Fault Tree Analysis [85]. Whereas FTA traces a top-level hazard to its lower-level causal events, ICPA traces a top-level state variable through the design to all components that influence it. ICPA uses a table structure similar to Failure Modes and Effects Analysis (FMEA) [50] to record these indirect control relationships. Agents along the trace path belong to the indirect control path and require further analysis of their relationships to the root variable. The indirect control relationships are then used in conjunction with the goal coverage strategy to define the safety subgoals and assign them to subsystems. The various aspects of the technique are demonstrated using an example distributed elevator system used in a graduate embedded systems course.

4.1.1 Motivation

The process of decomposing system safety goals into subgoals that are realizable by individual agents should provide:

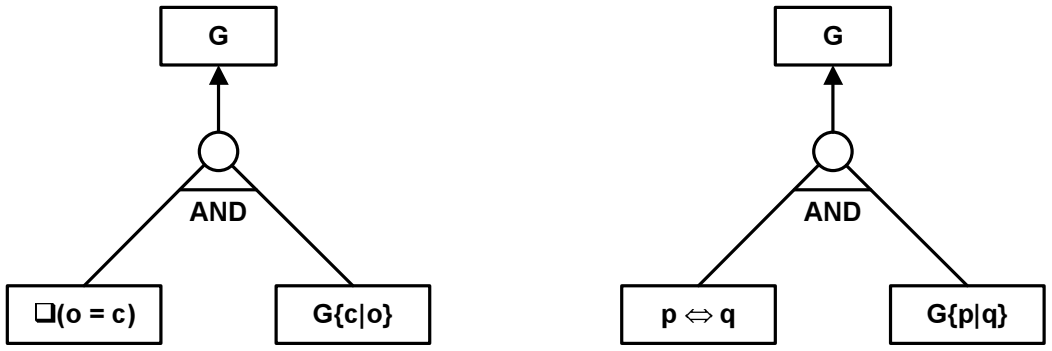
- systematic analysis of the system design to identify system components that directly

or indirectly control state variables in the parent goal

- documentation of the goal decomposition process
- documentation of critical assumptions of the safety goal decomposition
- documentation of restriction and redundancy in the decomposition.

Existing goal elaboration tactics are insufficient for system safety because although they provide support for comparing alternative decompositions [19][46], none provides a framework for guiding those comparisons and documenting the process. Unlike most functional requirements, which typically define the actions the system must take to accomplish some task, system safety goals often define what the system should do to avoid a hazardous state. In goal elaboration it is important to try to identify as many behaviors in the system as possible that could prevent the parent goal from being satisfied. In addition, the safety-critical system assurance process may require a safety case, defined by Bishop and Bloomfield as "a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment" [8]. In order to assure the system is safe, the process for decomposing system safety goals and assigning them to components must be deliberate, directed, and documented.

Another reason additional goal elaboration tactics are needed for system safety is because in general, the primary aim of goal elaboration is to define subgoals that meet the parent goal and are realizable by agents in the system [47]. Prior work in tactics for identifying missing goals [19][20] and resolving unrealizable goals [46][47] has focused on identifying subgoals that satisfy the parent goal without being restrictive and without employing redundancy. If emergence cannot be removed from the system, then a different strategy for goal elaboration is required. In the presence of emergence due to uncertainty,



(a) introduce actuation goal *on variable* (b) introduce actuation goal *on predicate*

Figure 4.1. Introduce accuracy/actuation goal elaboration tactic from [46]

it may be desirable to satisfy the goal with redundant subgoals. In the presence of emergence due to unrealizability, it may not be practical, or even possible, to satisfy the parent goal without restricting system behavior more than the parent goal requires. For safety goal elaboration, where consequences of not satisfying the goal are severe, restriction and redundancy in the decomposition may both be necessary.

This chapter addresses the following question posed in Chapter 1:

- How can full or partial decomposition of system safety goals be achieved?

4.1.2 Tactics for resolving goal unrealizability

ICPA relies upon the tactics developed by Letier and van Lamsweerde for identifying and resolving goal unrealizability [46][47]. In particular, tactics associated with resolving lack of monitorability and lack of control are used. An overview of the concept of realizability can be found in Section 2.3.2. This section provides an overview of three specific goal realizability tactics most-commonly used in ICPA: *Introduce Accuracy/Actuation Goal*, *Split Lack of Monitorability/Controllability by Chaining*, and *Split Lack of Monitorability/Controllability by Case*. All three tactics attempt to identify subgoals that are each real-

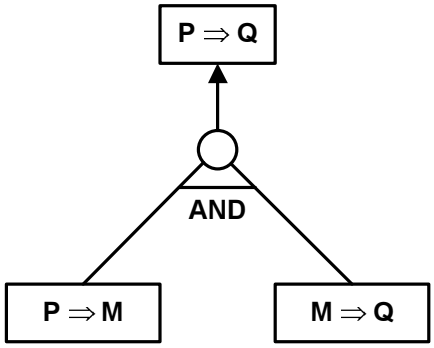


Figure 4.2. Split Lack of Monitorability/Controllability by Chaining goal elaboration tactic from [46]

izable by a single agent in the system by splitting the monitoring and control requirements for each goal.

The first tactic, *introduce accuracy/actuation goal*, is illustrated in Figure 4.1 from [46]. The basic idea of this tactic is to identify some other state variable in the system such that one function of a state variable in the parent goal is equivalent to another function of the other state variable in the system. In this case, the equivalence relationship between the two functions is one subgoal for the system (e.g., $\square(o = c)$ in Figure 4.1), and the function of the other state variable is the other subgoal (e.g., $G\{c|o\}$ in Figure 4.1). For lack of monitorability, the equivalent state variable function could be a sensor to monitor the state. For lack of control, the equivalent state variable would map to an actuator that changes the state.

The second tactic, *split lack of monitorability/controllability by chaining* [46], is shown in Figure 4.2. This tactic is applied to goals of the form $P \Rightarrow Q$, where P is the state variable or function of state variables to be monitored, and Q is the state variable or function of state variables to be controlled. The basic idea is to identify a sequence of subgoals of the same form that satisfy the parent goal that are each realizable by different agents. In other

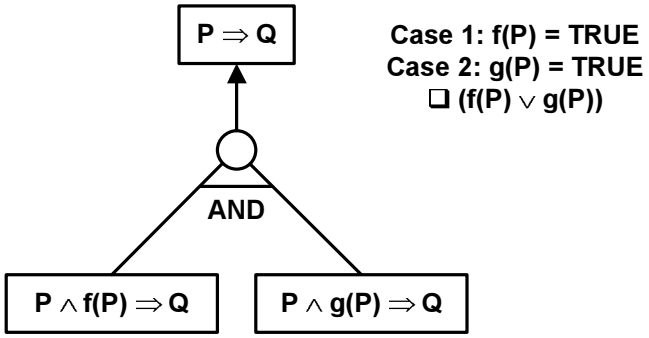


Figure 4.3. Split Lack of Monitorability/Controllability by Case goal elaboration tactic from [46]

words, this tactic employs coordination among agents that monitor and control different state variables.

The third tactic, *split lack of monitorability/controllability by case* [46], is shown in Figure 4.3. This tactic is applied when different control actions are required to satisfy the goal in different situations.

4.1.3 Scope

In this chapter, goals are defined as temporal, first-order logic expressions of system state, using the operators listed in Figure 2.5. In systems specified with these expressions, updates to a state variable cannot be observed by agents that monitor the variable until the subsequent state. Therefore, in order for a goal to be realizable, control actions in the goal cannot depend on future values of monitored state variables. Control actions can depend on present values of state variables, if the agent realizing the goal is also the agent controlling those state variables.

In addition, it is assumed that the system safety goals under review belong to a system for which a functional decomposition already exists. In other words, the system has already

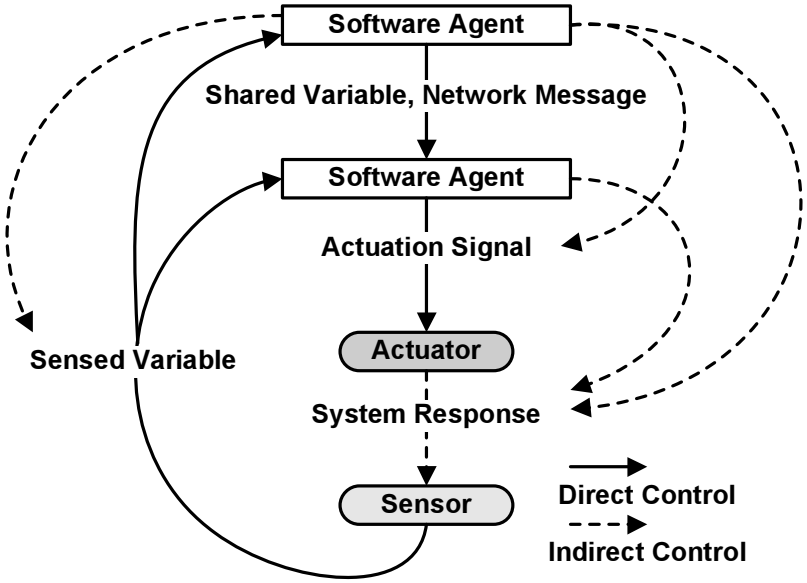


Figure 4.4. Indirect control paths

been divided into subsystems that have each been assigned a particular set of functional behaviors, and that communication between subsystems is known. The safety goal elaboration process may reveal the need to modify the functional decomposition, but this process begins with some baseline functional design.

The rest of this chapter is organized as follows: Section 4.2 introduces the notion of indirect control to the KAOS framework. Section 4.3 describes the format of an ICPA table. Section 4.4 discusses the procedure used to complete an ICPA. Finally, Section 4.5 defines the goal coverage strategies used in the ICPA process.

4.2 Indirect control

In the KAOS framework, goal realizability is driven by monitorability and controllability of system state variables [46][47]. Monitorability means the ability to observe the value of the state variable. Controllability means the ability to change the value of the state

variable. Furthermore, the KAOS framework specifies that only one agent may directly change the value of a given state variable [46]. Thus, only one agent can control any given state variable.

In composite systems, however, although one agent directly changes some state variable, other agents may also influence that change. The terms **direct control** and **indirect control** are introduced to distinguish between the ability to change and the ability to influence change in variables. Figure 4.4 shows the direct and indirect control relationships for an embedded system with sensing and actuation. Physical sources of indirect control may include hardware actuation, system dynamics, and environmental agents that change sensed state variables. The relationship between commands to an actuator and the actuator's effect on sensed system variables is indirect and defined by a function of the physical response of the system to the actuator command. The actuator does not change the sensed value directly. Rather, the sensor detects the sensed value from the system and environment.

Software agents directly control their own outputs, such as network messages, shared variables, and actuation signals, and indirectly control the consumers of those outputs (e.g., an actuator's response to an actuation signal). If one software agent $Ag1$ directly controls state variable Z based on input from another software agent $Ag2$, then $Ag2$ may indirectly control the system safety goal as well. If safety goal G restricts how Z can be controlled, then both agents $Ag1$ and $Ag2$ must be analyzed to see if they require a safety subgoal.

Figure 4.5 shows a partial design for a distributed elevator system. In this system, software agents *DoorController* and *DriveController* directly control the door motor and drive actuators. In addition, separate software agents for each hall call button and car call button, $HallButtonController_{f,d}$ and $CallButtonController_{f}$, process *Passenger* agent requests for floor f and direction d. Another software agent *DispatchController* contains a dispatch algorithm that tells the *DoorController* and *DriveController* agents the next scheduled des-

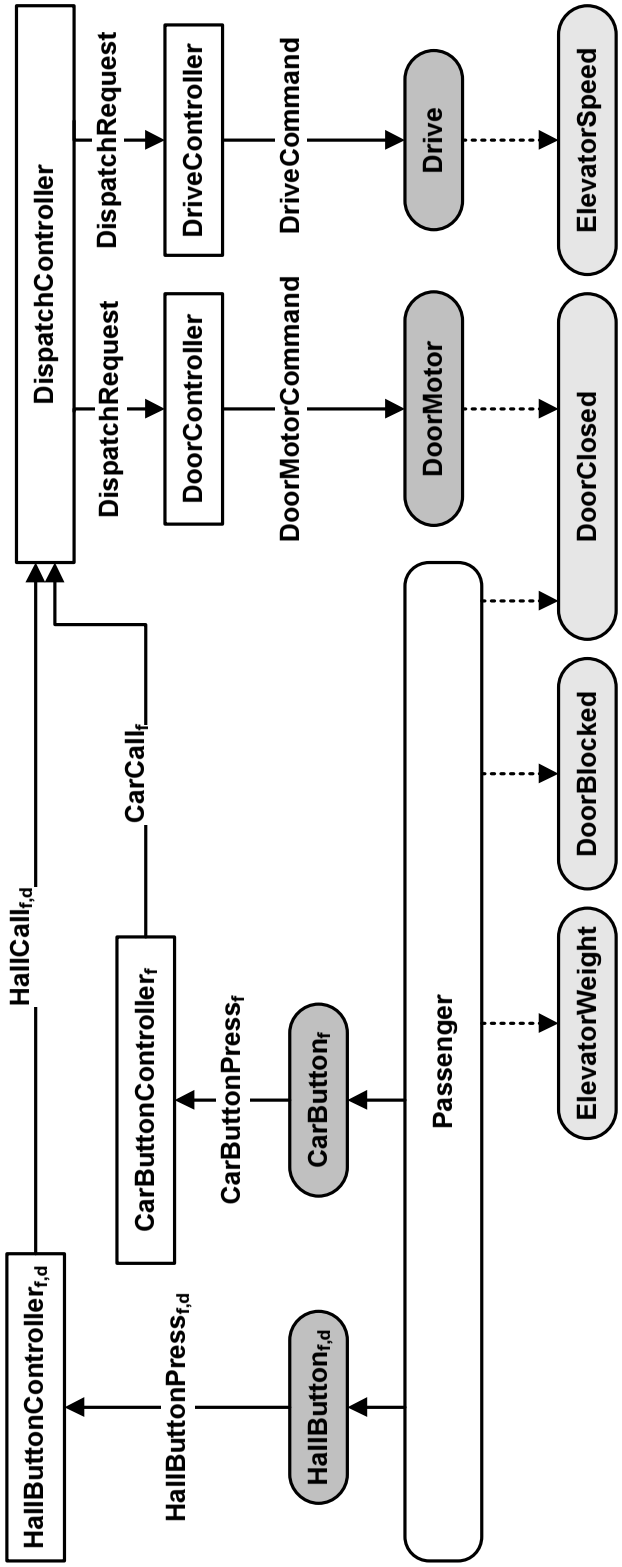


Figure 4.5. Partial design of a distributed elevator control system

Goal: Maintain[DriveStoppedWhenOverweight]
InformalDef: *If the elevator weight exceeds the weight threshold, then the elevator speed shall be STOPPED.*
FormalDef: ∀ ew: ElevatorWeight, es: ElevatorSpeed, wt: WeightThreshold
● (ew > wt) ⇒ IsStopped(es)

Figure 4.6. Goal restricting movement in an overweight elevator

tination. Sensors detect the elevator weight and speed, and whether the door is blocked or closed.

Consider the goal *Maintain[DriveStoppedWhenOverweight]*, shown in Figure 4.6, which restricts movement in the elevator when the elevator weight is over the limit for safe operation. In the distributed elevator system shown in Figure 4.5, *ElevatorWeight* is indirectly controlled by the *Passenger* agent. *ElevatorSpeed* is indirectly controlled by the *Drive* actuator, which is directly controlled via the actuation signal *DriveCommand* by the *DriveController* agent. *DriveController*, in turn, is influenced by the *DispatchController* agent via the network message *DispatchRequest*.

Unlike the notion of controllability in the KAOS framework, the notion of direct control in this thesis does not require strict controllability (i.e., more than one agent may directly control a system state variable). In networked distributed systems, multiple agents may send the same type of message. For example, in the distributed elevator system, hall button controllers on each floor may each generate a hall call message to other agents on a broadcast network. A goal that constrains control of hall call messages would apply to all hall call button controllers. Agent behaviors to generate the messages may be the same for all agents, such as different instantiations of the same car button controller, or they may differ by agent, depending on the system design.

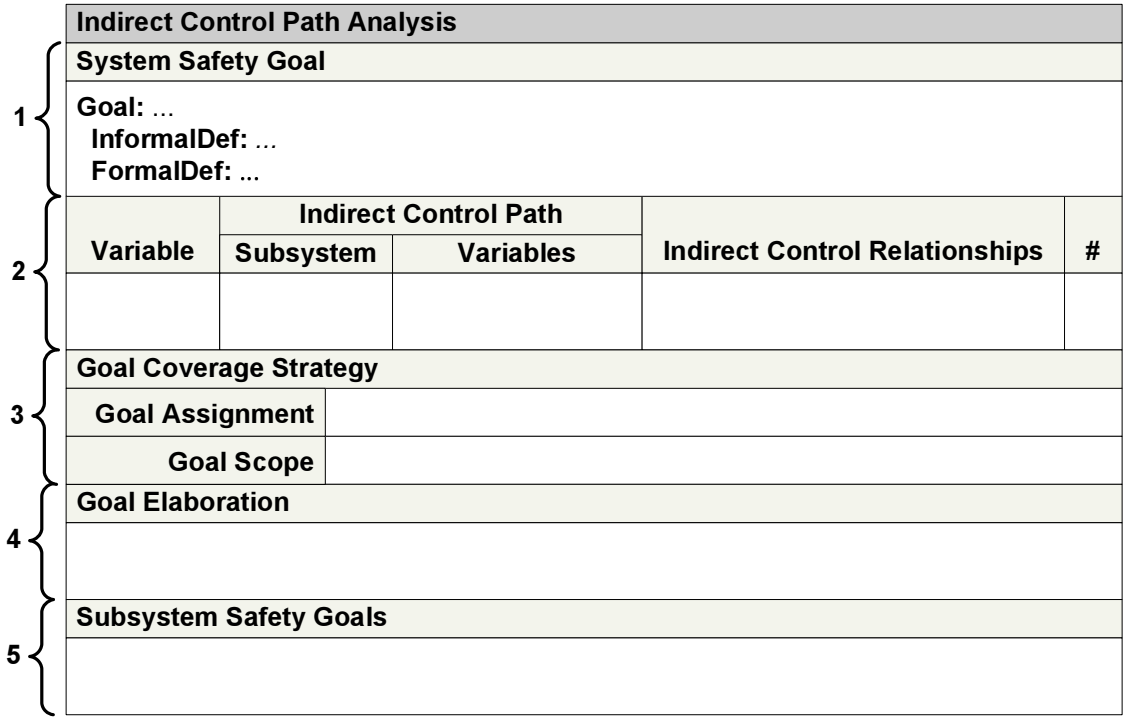


Figure 4.7. ICPA table format

4.3 ICPA format

One of the primary purposes of ICPA is documentation. It necessary to document not only the subgoals themselves, but also the process used to define them, the critical assumptions made during that process, and restriction or redundancy in the decomposition. Documentation of the safety goal elaboration process is necessary for several reasons. First, in an independent review of the safety goals and subgoals, perhaps during development of a safety case for the system [8], there needs to be a clear record of the critical goals and assumptions, as well as which subsystems and components were analyzed for indirect control during the goal elaboration process. In addition, as the development cycle progresses, changes to the design can be checked against the critical assumptions to determine if those changes impact the safety subgoals. Finally, it is important to document when subgoals

are more restrictive than the parent goal, as well as which subgoals satisfy a parent goal redundantly, to allow these decisions to be taken into account when future revisions to the requirements specification are made.

The basic layout of an ICPA table is shown in Figure 4.7. The first major section contains the system safety goal defined in the basic KAOS format [18]. The goal definition includes the goal name, an informal definition of the goal, and the formal definition in first order temporal logic.

The second major section includes fields for recording the state variables in the system safety goal, the indirect control paths for those variables, and the indirect control relationships along those paths. Indirect control paths include the subsystems that indirectly control the parent goal variable and the variables those subsystems control directly. Indirect control relationships are formal propositional logic expressions that relate the subsystem variables to the parent goal variables. Each relationship is numbered.

The third major section records the goal coverage strategy chosen for a particular goal. This includes fields for the goal assignment and goal scope. Goal coverage strategies are covered in more detail in Section 4.5.

The fourth major section contains the the indirect control relationships and restriction tactics used to define them. The goal relationships and restriction tactics listed here become critical assumptions of the subgoals. These critical assumptions, combined with the derived subgoals, form the decomposition of the parent goal.

The final section lists the subgoals themselves, using the same format as the parent goal in the first section.

4.4 ICPA procedure

The other primary purpose of ICPA is to guide safety goal elaboration in a systematic way that attempts to uncover all hidden dependencies related to the state variables in the parent goal.

For functional requirements, the purpose of goal elaboration is usually creation of behavior. Functional goal elaboration is used to develop a system design that performs the required actions. In contrast, the aim of safety goal elaboration is usually restriction. As such, goal elaboration is used to to identify which system components that may affect the system safety goal, and identify subgoals for some of those components to prevent the system from entering a hazardous state. In some situations new components are required in order to satisfy the system safety goal. However, in most situations safety goal elaboration occurs once a functional decomposition has been established, and is about constraint, rather than creation. A process for functional goal elaboration may entail brainstorming about types of components that can be included to perform the required task, whereas a systematic process for safety goal elaboration requires directed examination of all subsystems that may influence the behavior of state variables in the system safety goal.

In addition, the dependencies among the state variables produced by the indirect control sources and the state variables contained in the parent goal must be formally defined. These dependencies serve as the starting point for applying tactics for pattern-based goal elaboration [19][20] and resolving unrealizable goals [46][47]. In order to ensure the parent goal is satisfied by the derived tactics, it is important to uncover all dependencies in the system. This becomes more difficult to do as the system design gets more complex. Although a directed search may not guarantee all such dependencies are uncovered, it may uncover more dependencies than an undirected search.

The ICPA approach to safety goal elaboration is fairly straightforward. First, indirect

control sources are identified for each state variable in the parent goal, using the system's functional decomposition and subsystem inputs and outputs. Next, the relationships among the state variables in the parent goal and the state variables controlled by agents in the indirect control path are formally defined. The following sections explain these steps in greater detail.

4.4.1 Identifying indirect control sources

The first step in ICPA is identifying the direct and indirect control sources of state variables in the formally specified parent goal. To do this, the functional decomposition, including communication paths between components, is reviewed. Depending on the stage of system development, this decomposition may be documented in the functional requirements or the design specification. A graphical illustration of the indirect control paths can be useful for this this step, such as the one shown in Figure 4.5. However, in complex systems a complete diagram may be difficult to construct. In these systems, other documentation of the system design, such as input/output lists for each subsystem, may be used.

The first stop along the indirect control path is identifying the subsystems or components that directly produce the state variable in the system. If the state variable is a command to an actuator, then the source of direct control is usually some software agent. In Figure 4.5, the subsystem *DriveConroller* directly controls the signal *DriveCommand* to the *Drive* actuator. In some situations, an environmental agent may have direct control of actuator commands. In the same elevator, the *Passenger* agent directly controls the $CarButton_f$ and $DoorButton_{f,d}$ actuators. If the state variable in the parent goal is a sensed value in the system, then there is no source of direct control. The nearest sources of indirect control are the actuators that interact with the environment to change the system state variable detected by the sensor, and those are directly controlled by set points. Once the source of direct control

Goal: Maintain[DoorClosedOrElevatorStopped]
InformalDef: *At all times the door shall be closed or the elevator speed shall be STOPPED*
FormalDef: ∀ dc: DoorClosed, es: ElevatorSpeed
❑ (dc ∨ IsStopped(es))

Figure 4.8. Goal restricting door position and elevator movement in a distributed elevator system

is known, input sources for that agent are examined, and the process repeats.

Consider the indirect control paths for the distributed elevator system shown in Figure 4.5. A safety goal that restricts state variables *DoorClosed* and *ElevatorSpeed* is shown in Figure 4.8. In the system these state variables are directly controlled by motion detection sensors. However, each is indirectly controlled by agents in the system.

The control path of *ElevatorSpeed* includes one branch, which contains *Drive*, *DriveController*, *DispatchController*, $CarButtonController_f$, and $HallButtonController_{f,d}$. *Drive* is an actuator that controls the physical elevator speed, which is then detected by the elevator speed sensor to generate the *ElevatorSpeed* state variable. *DriveController* is a software agent that commands the *Drive* to a certain speed. *DispatchController* is another agent that tells *DriveController* the desired destination of the elevator. $CarButtonController_f$ and $HallButtonController_{f,d}$ are software agents that tell *DispatchController* when a button is pressed to request a specific destination.

The control path of *DoorClosed* is branched. The first branch contains *DoorMotor*, *DoorMotorController*, and *DispatchController*. The second branch is the *Passenger*. *DoorMotor* is an actuator that controls door position, which includes the closed position detected by a sensor to generate the *DoorClosed* state variable. *DoorMotorController* is

a software agent that commands the *DoorMotor* to a certain speed. *DispatchController* is another agent that tells *DoorMotorController* the desired destination of the elevator. $CarButtonController_f$ and $HallButtonController_{f,d}$ are software agents that tell *DispatchController* when a button is pressed to request a specific destination. *Passenger* is an environmental agent who interacts with the elevator control system. *Passenger* can inhibit *DoorClosed* from being *TRUE* by physically blocking the doors.

Similar to fault paths in FTA, indirect control paths in ICPA may have many branches. That is, each source along an indirect control path of a parent goal variable may have multiple sub-paths of control. Likewise, one particular path may include many sources. Depending on the system design, the number of indirect control sources may be quite numerous or circular. To keep this analysis manageable, the second and third steps of the ICPA, described in Sections 4.4.2 and 4.4.3 below, can be performed at each level in the indirect control path. If a set of subgoals for that level is found to satisfy the parent goal, then further analysis of the indirect control path is only necessary for identifying redundant subgoal sets. If not, then the process is repeated at the next level along the indirect control path, or additional components may be needed to satisfy the goal.

4.4.2 Defining indirect control relationships

Once indirect control sources have been identified, their relationship to the original variable must be defined in such a way that the general agent-based elaboration tactic *Introduce Accuracy/Actuation Goal*, and *Split Lack of Monitorability/Controllability by Chaining* from [46] can be applied. As described in Section 4.1.2 if these relationships can be defined in the form $(o = c)$ or $(p \Leftrightarrow q)$, then the goal G can be defined as $G\{o|c\}$ or $G\{p|q\}$. In other words, goals can be defined if the dependencies between state variables define functions that relate the variable in the parent goal to the indirect control variables. If the

parent goal is of the form ($P \Rightarrow Q$) and a relationship can be defined such that ($P \Rightarrow M$), a subgoal could be defined ($M \Rightarrow Q$). In this situation, the entailment in the original goal is achieved by a sequence of entailments in the subgoals.

For paths with a single branch, the indirect control relationship is defined between state variables controlled by each pair of agents along the path and state variables contained in the parent goal. For paths with multiple branches, the indirect control relationship is defined among state variables controlled by agents at a given level in the indirect control path and state variables contained in the parent goal.

Indirect control relationships for the first indirect control path level of variables *DoorClosed* and *ElevatorSpeed* in the goal *Maintain[DoorClosedOrElevatorStopped]* from Figure 4.8 are presented in Tables 4.1 and 4.2. The indirect control path for *ElevatorSpeed* has one branch. Because *ElevatorSpeed* is a sensed value that is not directly changed by any agent in the system, the closest indirect control source, *Drive*, is examined first. The relationship between sensed value *ElevatorSpeed* and the actuated value *DriveSpeed* indicates that when the drive is stopped the elevator will be stopped also:

$$IsStopped(DriveSpeed) \Leftrightarrow IsStopped(ElevatorSpeed) \tag{4.1}$$

DriveSpeed, however, is not directly controlled by any software agent in the system, either. Instead, it is indirectly controlled by *DriveController* using the actuation signal *DriveCommand*. A drive that is commanded to stop will do so, but only after some delay:

$$\bullet\blacksquare_{<MaxStopDelay}(DriveCommand = \text{`STOP'}) \Rightarrow IsStopped(DriveSpeed) \tag{4.2}$$

$$\begin{aligned}&(\bullet\neg IsStopped(DriveSpeed) \wedge \bullet\blacklozenge_{<MinStopDelay}@(DriveCommand = \text{`STOP'}))\\ &\Rightarrow \neg IsStopped(DriveSpeed)\end{aligned} \tag{4.3}$$

Table 4.1. Indirect control paths for goal Maintain[DoorClosedOrElevatorStopped] (1 of 2)

System Safety Goal				
Goal: Maintain[DoorClosedOrElevatorStopped] **InformalDef:** *At all times the door shall be closed or the elevator speed shall be STOPPED.* **FormalDef:** dc: DoorClosed, es: ElevatorSpeed ❑ (dc ∨ IsStopped(es))				
Variable	**Indirect Control Path**		**Indirect Control Relationships**	**#**
	Subsystem	**Variables**		
dc	DoorController, DoorMotor	dmc: DoorMotorCommand maxcd: MaxCloseDelay mincd: MinCloseDelay maxod: MaxOpenDelay minod: MinOpenDelay dms: DoorMotorSpeed ssd: SingleStateDuration	$S_0 \models \neg dc \wedge dmc = \text{`OPEN'}$ % In initial state, door is OPEN and commanded OPEN	01
			$(\bullet dc \wedge \bullet (dmc = \text{`CLOSE'})) \Rightarrow dc$ % Closed door that is commanded CLOSE remains closed	02
			$\bullet \blacksquare_{< maxcd} (\neg db \wedge (dmc = \text{`CLOSE'})) \Rightarrow dc$ % Unblocked door commanded CLOSE for *maxcd* will be closed	03
			$(\bullet \neg dc \wedge \bullet \blacklozenge_{< mincd} @(dmc = \text{`CLOSE'}) \Rightarrow \neg dc$ % Unclosed door whose command switched to CLOSE from % OPEN within *mincd* will not be closed	04
			$(\bullet \neg dc \wedge \bullet (dmc = \text{`OPEN'})) \Rightarrow \neg dc$ % Unclosed door commanded OPEN remains unclosed	05
			$\bullet \blacksquare_{< maxod} (dmc = \text{`OPEN'}) \Rightarrow \neg dc$ % Door commanded OPEN for *maxod* will be unclosed	06
			$(\bullet dc \wedge \bullet \blacklozenge_{< minod} @(dmc = \text{`OPEN'}) \Rightarrow dc$ % Closed door that is closed and whose command switched to OPEN from CLOSE within duration *minod* will be closed	07
			maxcd > mincd >> ssd % CLOSE delays are greater than state	08
			maxod > minod >> ssd % OPEN delays are greater than state	09

Table 4.2. Indirect control paths for goal Maintain[DoorClosedOrElevatorStopped] (2 of 2)

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
dc	Passenger	db: DoorBlocked	● db ⇒ dmc = `OPEN' % if the door is blocked, the door shall be commanded OPEN	10
			● db ⇒ ¬ dc % if the door is blocked, the door shall not be closed	11
es	DriveController, Drive	drc: DriveCommand maxsd: MaxStopDelay minsd: MinStopDelay maxgd: MaxGoDelay mingd: MinGoDelay drs: DriveSpeed ssd: SingleStateDuration	$S_0 \models$ IsStopped(es) ∧ drc = `STOP' % In initial state, elevator stopped and drive commanded STOP	12
			IsStopped(drs) ⇔ IsStopped(es) % If the drive is stopped, the elevator is stopped, and vice versa	13
			(● IsStopped(drs) ∧ ● (drc = `STOP')) ⇒ IsStopped(drs) % Stopped drive commanded STOP remains stopped	14
			● ■ $_{< maxsd}$(drc = `STOP') ⇒ IsStopped(drs) % Drive commanded STOP for maxsd will be stopped	15
			(● ¬ IsStopped(drs) ∧ ● ♦ $_{< minsd}$ @(drc = `STOP') ⇒ ¬ IsStopped(drs) % Unstopped drive whose command switched to STOP from GO % within duration minsd remains unstopped	16
			(● ¬ IsStopped(drs) ∧ ● (drc = `GO')) ⇒ ¬ IsStopped(drs) % Unstopped drive commanded GO remains unstopped	17
			● ■ $_{< maxgd}$ (drc = `GO') ⇒ ¬ IsStopped(drs) % Drive commanded GO for maxgd will be unstopped	18
			(● IsStopped(drs) ∧ ● ♦ $_{< mingd}$ @(drc = `GO') ⇒ IsStopped(drs) % Stopped drive whose command switched to GO from STOP % within duration mingd remains stopped	19
			maxsd > minsd >> ssd % STOP delays are greater than state	20
			maxgd > mingd >> ssd % GO delays are greater than state	21

Likewise, a drive that is commanded to go will do so, but only after some delay:

$$\bullet\blacksquare_{<MaxGoDelay}(DriveCommand = \text{`GO'}) \Rightarrow \neg IsStopped(DriveSpeed) \tag{4.4}$$

$$\begin{aligned}&(\bullet IsStopped(DriveSpeed) \wedge \bullet\blacklozenge_{<MinGoDelay}@(DriveCommand = \text{`GO'}))\\ &\quad \Rightarrow IsStopped(DriveSpeed)\end{aligned} \tag{4.5}$$

Variables with multiple branches require relationship among branches to be defined as well. Sometimes, the branches represent independent control paths (i.e., one path traversed at a time). In these situations each branch can be evaluated as if it were a single branch path. Sometimes, however, the control paths in multiple branches are interdependent. In the distributed elevator system of Figure 4.5, the door controller commanding the door motor to close the doors should eventually set the *DoorClosed* sensor to *'TRUE'*. However, objects or passengers blocking the doors can physically prevent the door from being closed, as represented by the following relationship:

$$\bullet DoorBlocked \Rightarrow \neg DoorClosed \tag{4.6}$$

The relationship between blocking agents and the door motor is also constrained by a related safety goal that requires a door reversal if the door is blocked.

$$\bullet DoorBlocked \Rightarrow DoorMotorCommand = \text{`OPEN'} \tag{4.7}$$

As a design choice for this system, this safety goal is given priority over the safety goal *Maintain[DoorClosedOrElevatorStopped]*. In other words, if the elevator enters a state where the door is blocked and the elevator is moving, the preferred action is to open the doors.

As a result, a door that is commanded to close will do so after some delay, provided no agents or objects are blocking the door:

$$\bullet\blacksquare_{<MaxCloseDelay}(\neg DoorBlocked \wedge (DoorMotorCommand = \text{`CLOSE'})) \Rightarrow DoorClosed \tag{4.8}$$

$$(\bullet\neg DoorClosed \wedge \bullet\blacklozenge_{<MinCloseDelay}@(DoorMotorCommand = \text{`CLOSE'})) \Rightarrow \neg DoorClosed \tag{4.9}$$

However, a door that is commanded to open will no longer be closed after some delay, whether or not any agent or object is blocking the door:

$$\bullet\blacksquare_{<MaxOpenDelay}(DoorMotorCommand = \text{`OPEN'}) \Rightarrow \neg DoorClosed \tag{4.10}$$

$$(\bullet DoorClosed \wedge \bullet\blacklozenge_{<MinOpenDelay}@(DoorMotorCommand = \text{`OPEN'})) \Rightarrow DoorClosed \tag{4.11}$$

4.4.3 Applying elaboration tactics and goal coverage strategies

The next step in ICPA is to use the indirect control relationships defined in the previous step to identify subgoals for subsystems that satisfy the parent goal. This is done by application of the goal realizability tactics defined by Letier and van Lamsweerde [46][47], and by choosing and applying a particular goal coverage strategy. Goal coverage strategies are discussed in more detail in Section 4.5. An overview of the concept of goal realizability can be found in Section 2.3.2. That section provides descriptions of two specific goal realizability tactics most-commonly used in ICPA: *Introduce Accuracy/Actuation Goal*, and *Split Lack of Monitorability/Controllability by Chaining*. Further detail on those and other

tactics can be found in [46][47].

As mentioned in Section 4.4.1, a state variable in a system safety goal may be large in breadth (number of branches) or depth (number of stops along each branch). ICPA approaches goal coverage by starting from the indirect control level nearest the parent goal variable, and working outward along the branches. When a particular level of indirect control is branched, sources on all branches must be included in the examination. If a goal decomposition that satisfies the parent goal can be obtained, further analysis along the control path can be halted. However, if redundant goal coverage is required, the analysis should continue outward along the branches from the parent goal variable.

Table 4.3 records the goal coverage strategy and goal elaboration for the indirect control paths of the goal *Maintain[DoorClosedOrElevatorStopped]* from Tables 4.1 and 4.2. The goal coverage strategy used to satisfy the parent goal includes a goal assignment of shared responsibility between *DoorController* and *DriveController*, and a restrictive goal scope (worst-case actuation delays are exploited to ensure the goal is satisfied). Further details about the goal coverage strategy used in this example can be found in Section 4.5.

The goal elaboration section of the table includes the specific goal realizability tactics and indirect control relationships used to determine the missing subgoals. These indirect control relationships become critical assumptions that must be ensured in the system, in addition to the newly defined subgoals, in order to satisfy the parent goal. For the goal *Maintain[DoorClosedOrElevatorStopped]*, the first critical assumption used in goal elaboration is the specification of the initial system state. That allows the goal, which requires control of two different actuators, to to be made realizable by applying the *split lack of monitorability/controllability by case* tactic shown in Figure 4.3. The first case is the initial state, which is defined by critical assumptions (1) and (10). The second case includes all subsequent states. In this second case, separate goals restrict control actions performed on

Table 4.3. Goal Elaboration for goal Maintain[DoorClosedOrElevatorStopped]

System Safety Goal	
Goal: Maintain[DoorClosedOrElevatorStopped] **InformalDef:** *At all times the door shall be closed or the elevator speed shall be STOPPED.* **FormalDef:** dc: DoorClosed, es: ElevatorSpeed ❑ (dc ∨ IsStopped(es))	
Goal Coverage Strategy	
Goal Assignment	Shared Responsibility (DoorController & DriveController)
Goal Scope	Restrictive (Assumes worst-case actuator response times; real response may be slower.)
Goal Elaboration	
❑ (dc ∨ IsStopped(es)) (●¬ dc ⇒ IsStopped(es)) ∧ (●¬ IsStopped(es) ⇒ dc)	Indirect Control Relationships 01, 12 – Goal satisfied in initial state, Split lack of monitorability/control by case
●¬ IsStopped(es) ⇒ dc ((●¬ IsStopped(es) ∨ ● (drc = `GO')) ∧ (●¬ db )) ⇒ (dmc = `CLOSE')	07, 09 – Minimum delay to open door 10 – Door reversal safety goal 13 – Introduce accuracy goal tactic 02 – Remain closed w/ CLOSE command 19, 21 – Minimum delay to move elevator
●¬ dc ⇒ IsStopped(es) (●¬ dc ∨ (● dmc = `OPEN')) ⇒ (drc = `STOP')	07, 09 – Minimum delay to open door 13 – Introduce actuation goal tactic 14 – Remain stopped w/ STOP command 19, 21 – Minimum delay to move elevator

DriveController and control actions performed on *DoorController*. This elaboration also relies on actuation delays for both *DoorMotor* and *Drive* being much greater than a single state.

Depending on the tactic used in goal elaboration, the goal elaboration field of an ICPA may contain a proof showing how the subgoals and critical assumptions entail the parent goal. In particular, the goal realizability tactics *Introduce Accuracy/Actuation Goal*, and *Split Lack of Monitorability/Controllability by Chaining* from [46][47] are amenable to this approach.

If other tactics are used, particularly when the goal assignment requires coordinated control, the goal elaboration field may not contain a formal proof. ICPA uses formal specification of system goals to guide goal elaboration, but the intent of formal specification is not necessarily to provide a formal proof of all subgoal decompositions. Rather, the primary intent is to ensure that the safety goals and critical assumptions are precisely defined and documented. The goal elaboration field should contain a list of the critical assumptions and elaboration tactics used to define the subgoals. The parent goals could be verified against the subgoals and indirect control relationships with model-checking [17], or monitored at run-time along with the subgoals to detect when they become invalid.

4.4.4 Iteration and completion

Like the system development process in general, the ICPA process is not purely linear. The products of previous stages are revisited during later stages and modified as necessary. For example, a goal coverage strategy may be chosen or changed when goal realizability tactics are applied.

Defining indirect control sources and the relationships among them is somewhat open-ended. Each state variable in the parent goal has some indirect control path, with one or

more branches and one or more stops along the path. At a minimum, all indirect control sources at the stops closest to the parent goal along all branches should be analyzed. For goal redundancy, more stops along the path may be included in the analysis.

Indirect control relationships are gleaned from examination of the functional requirements, the control architecture, the design specification, and the implementation. Therefore, the state of these artifacts during the ICPA influences how many indirect control relationships are defined. In general, the intent is to identify all dependencies among the state variables identified as indirect control sources or included in the parent goal. As the development cycle progresses, newly defined dependencies are added to the table of indirect control relationships.

4.5 Goal coverage strategies

In any system design, there are often alternative subgoal decompositions that will satisfy a given parent goal. Subgoals may or may not be required for every indirect control agent in order to satisfy the parent goal. The system safety goal may be satisfied by one agent in the system, or may require coordination among agents. The subgoals may exactly meet the system safety goal, or may be more restrictive of functional behavior. Sometimes, redundant goal coverage may be desirable.

A **goal coverage strategy** is a plan for allocating subgoals to ensure that a high-level goal is met. Each goal coverage strategy is defined by goal assignment and goal scope. In this section, different classifications of goal assignment and goal scope are defined. Section 4.5.1 describes single responsibility, redundant responsibility, and shared responsibility goal assignments. In Section 4.5.2, nonrestrictive and restrictive goal scope are defined. In both sections, subgoal patterns for common hazard reduction techniques are also presented.

Goal: Maintain[ElevatorBelowHoistwayUpperLimit]
InformalDef: *The top of the elevator shall never exceed the upper limit of the hoistway.*
FormalDef: ∀ etp: ElevatorTopPosition, hul: HoistwayUpperLimit
❑ (etp ≤ hul)

Figure 4.9. Goal restricting elevator position in the hoistway

4.5.1 Goal assignment

Goal assignment defines which indirect control sources have subgoals and how those subgoals relate to each other. It may be driven by physical limitations of the system (e.g., actuation delays described in Section 4.4.2). It may also be influenced by potential loss of monitorability and controllability by agents in the system. The three categories of goal assignment presented in this section are *single responsibility*, *redundant responsibility*, and *shared responsibility*.

Single responsibility. In the base case for general goal elaboration, the safety goal is met by assigning one or more subgoals to a single agent. For system safety, a single responsibility goal assignment facilitates isolation of safety-critical behaviors from other non-critical components. It also allows more rigorous (and expensive) development processes to be applied to those isolated, fewer agents. The agent responsible for meeting the goal may also be responsible for other, non-critical functionality. Alternately, an agent's behaviors may be limited to simply satisfying the safety goal (e.g., a safety monitor).

Consider a safety goal in a distributed elevator system that restricts elevator position relative to the end of the hoistway, as shown in Figure 4.9. In the elevator control system depicted in Figure 4.5, the goal could be met by requiring the drive controller to stop the elevator before the hoistway limit is reached, as shown in Figure 4.10.

Goal: Achieve[StopBeforeHoistwayUpperLimit]
InformalDef: *If the elevator nears the upper hoistway limit, then the drive shall be stopped.*
FormalDef: ∀ etp: ElevatorTopPosition, hul: HoistwayUpperLimit, msd: MaxStoppingDistance, drc: DriveCommand
●(etp ≥ (hul - msd)) ⇒ drc = 'STOP')

Figure 4.10. Goal restricting elevator drive movement in the hoistway

Single responsibility requires an agent or group of agents to monitor or control all state variables in the defined subgoals.

Redundant responsibility. A single responsibility strategy may be desirable for reducing development costs, but it also provides a single point of failure for the safety goal. Functional redundancy is a common strategy for fault tolerance in which different agents perform the same set of required functions [50]. This redundant functionality may be identical, such as duplicate networks for tolerating dropped messages, or different, such as a backup that provides a minimal set of functions when the primary fails.

Goal redundancy is achieved by assigning primary responsibility to one group of agents; secondary, to another group. If at least one group of agents satisfy their subgoals, the parent goal will also be satisfied. Redundant subgoals may have the same goal scope, or may have varying degrees of restriction compared to the original system safety goal. If subgoals vary in restriction, agents with primary responsibility have the most restrictive subgoals and agents with secondary responsibility have less restrictive subgoals (i.e., normal behavior has a greater safety margin than emergency backup behavior). Goal scope is discussed in Section 4.5.2.

In the elevator system, drive controller reliability may be too low or too unmeasurable to ensure the safety goal is met, particularly for complex software control. Physical compo-

Goal: Achieve[EmergencyStopBeforeHoistwayUpperLimit]
InformalDef: *If the elevator nears the upper hoistway limit, then the emergency brake shall be applied.*
FormalDef: ∀ etp: ElevatorTopPosition, hul: HoistwayUpperLimit, mebd: MaxEmergencyBrakingDistance, eb: EmergencyBrake
●(etp ≥ (hul - mebd)) ⇒ eb = 'APPLIED')

Figure 4.11. Goal requiring emergency braking to avoid exceeding the hoistway limit

nent reliability, such as a physical emergency brake trigger, is better known. A goal for an emergency brake agent in the system is shown in Figure 4.11.

However, relying on the emergency brake alone to meet the safety goal is also undesirable because of equipment wear and harm to passengers with emergency braking stops. By assigning primary responsibility to the elevator drive controller and secondary responsibility to the emergency brake, the safety goal may be more reliably met while largely avoiding application of the physical emergency brake.

Shared responsibility. Sometimes a safety goal may require coordination among agents if physical system dynamics limit variable controllability and observability. In shared responsibility, two or more agents have subgoals that must be met in order to meet the parent goal.

The goal *Maintain[DoorClosedOrElevatorStopped]* defined in Section 4.4.2 cannot be assigned to the drive controller or door controller alone because of physical actuation delays and the inability to monitor and control in the same state. In order to achieve the goal, which prohibits states in which the door is open and the elevator is moving, a single agent must have the ability to control both the door motor and the drive. If an agent can only monitor

Goal: Achieve[CloseDoorWhenElevatorMoving]
InformalDef: *If the elevator is moving,*
then the door shall be commanded to 'CLOSE'.
FormalDef: ∀ dmc: DoorMotorCommand, es: ElevatorSpeed,
db: DoorBlocked
(●¬IsStopped(es) ∧ ●¬ db) ⇒ (dmc = 'CLOSE')

Figure 4.12. Goal restricting elevator door movement

the behavior of one actuator while controlling the other, the goal cannot be satisfied for two reasons. First, monitored values are only known for the previous state. Any goal that requires a value to be monitored in the same state in which the control action occurs is not realizable. Second, control actions by the agent are subject to actuator delays.

Suppose the door controller alone is given responsibility for the goal with the subgoal defined in Figure 4.12. Operationalization of this goal prohibits opening the door while the elevator is in motion and prescribes closing the door if the elevator moves, except when a passenger is blocking the doors. If the drive controller activates the drive motor while the doors are already open, the safety goal will be violated during the actuation delay required to close the doors.

Now, suppose another subgoal is defined to prevent the drive controller from moving the elevator while the doors are open, as shown in Figure 4.13. Operationalization of this goal prohibits moving the drive while the doors are open and prescribes stopping the drive if the elevator moves.

Even though the behavior of both controllers is restricted, the parent safety goal may be violated when the elevator is stopped and the doors are closed, if the door controller attempts to open the doors at the same time as the drive controller attempts to move the elevator. In this situation, it is important to monitor both the sensed value of the parent goal

Goal: Achieve[StopElevatorWhenDoorOpen]
InformalDef: *If the door is open, then the drive shall be commanded to 'STOP'.*
FormalDef: ∀ drc: DriveCommand, dc: DoorClosed
(●¬dc) ⇒ (drc = 'STOP')

Figure 4.13. Goal restricting elevator drive movement

and its indirect control actuation source.

Table 4.4 shows the subgoals for the door controller and drive controller defined during the ICPA of the parent goal *Maintain[DriveStoppedWhenOverweight]*. The door controller monitors both elevator motion and drive commands. The drive controller monitors both the door closed sensor and the drive actuator commands. If a) the initial state of the system is known and does does not violate the system safety goal, b) monitored values are delayed one state, and c) the physical actuation delays are much greater than a single state, then each controller will be able to cancel its own actuation command if it observes the other controller commanding actuation, before the doors or drive have actually begun to move.

Sometimes physical actuation and network delays are insufficient for ensuring the goal is met. An *interlock* [50] is a common solution for enforcing sequencing of coordinated actions in systems. Suppose a safety goal coordinating two actions takes the form ❑ ($A \vee B$)), where A is indirectly controlled by agent agA and B, by agB. The basic patterns of the primary subgoals are:

$$\bullet\neg B \Rightarrow A \tag{4.12}$$

$$\bullet\neg A \Rightarrow B \tag{4.13}$$

Before negating A, agA must set a variable L_A and check that agB's interlock variable L_B is not set. This is similar to a mutex or semaphore [24] used in software programs to

Table 4.4. Subgoals of Maintain[DoorClosedOrElevatorStopped] for DoorController and DriveController]

System Safety Goal
Goal: Maintain[DoorClosedOrElevatorStopped] **InformalDef:** *At all times the door shall be closed or the elevator speed shall be STOPPED.* **FormalDef:** dc: DoorClosed, es: ElevatorSpeed $\Box$ (dc $\vee$ IsStopped(es))
Subsystem Safety Goals
Subsystem: DoorController **Controls:** DoorMotorCommand **Observes:** ElevatorSpeed, DriveCommand, DoorBlocked **Goal:** Achieve[CloseDoorWhenElevatorMovingOrMoved] **InformalDef:** *If the door is not blocked and the elevator a) is moving or b) has been commanded to move, then the door shall be commanded to CLOSE.* FormalDef: $\forall$ drc: DriveCommand, dmc: DoorMotorCommand, es: ElevatorSpeed ((● $\neg$ IsStopped(es) $\vee$ (● drc = `GO')) $\wedge$ (● $\neg$ db )) $\Rightarrow$ (dmc = `CLOSE')
Subsystem: DriveController **Controls:** DriveCommand **Observes:** DoorClosed, DoorMotorCommand **Goal:** Achieve[StopElevatorWhenDoorOpenOrOpened] **InformalDef:** *If the doors a) are not closed or b) have been commanded open, then the drive shall be commanded to STOP.* FormalDef: $\forall$ drc: DriveCommand, dmc: DoorMotorCommand, dc: DoorClosed (● $\neg$ dc $\vee$ (● dmc = `OPEN')) $\Rightarrow$ (drc = `STOP')

coordinate the use of shared resources. Basic patterns of subgoals using an interlock are:

$$\bullet(\neg L_A \vee L_B) \Rightarrow A \tag{4.14}$$

$$\bullet(\neg L_B \vee L_A) \Rightarrow B \tag{4.15}$$

Now suppose A and B have actuation delays, where A_1 causes A to be set after some delay, and A_2 causes A to be unset after some delay. The indirect control relationships for setting and unsetting A are defined as:

$$\bullet\blacksquare_{<MaxDelayA} A_1 \Rightarrow A \tag{4.16}$$

$$(\bullet\neg A \wedge \bullet\blacklozenge_{<MinDelayA} @A_1) \Rightarrow \neg A \tag{4.17}$$

$$\bullet\blacksquare_{<MaxDelay\neg A} A_2 \Rightarrow \neg A \tag{4.18}$$

$$(\bullet A \wedge \bullet\blacklozenge_{<MinDelay\neg A} @A_2) \Rightarrow A \tag{4.19}$$

$$\square\neg(A_1 \wedge A_2) \tag{4.20}$$

If all variables shared between agents also have communication delays, the new subgoals for agA are:

$$\bullet\blacklozenge_{<MinComDelay}(\neg B \vee B_2) \Rightarrow A_1 \wedge \neg A_2 \tag{4.21}$$

$$\bullet\blacklozenge_{<MinComDelay}(\neg L_A \vee L_B) \Rightarrow A_1 \wedge \neg A_2 \tag{4.22}$$

$$\square(MinComDelay < MaxDelay\neg A) \tag{4.23}$$

The subgoals for agB are analogous.

A *lockout* coordinates enforcement of safety goals by prohibiting an action from occur-

ring [50]. For example, a bus guardian [33] is used to prevent faulty nodes on a network from interfering with communication by others. In time-triggered networks a bus guardian will enable transmission access only during the node's allotted time slot [81]. Suppose a safety goal takes the form $\bullet\blacklozenge_{<T}\ (D \Rightarrow \neg C)$, where C is indirectly controlled by agent agA and D is observed by agent agA. The control relationship of C by agA is defined by:

$$\bullet A \Rightarrow C \tag{4.24}$$

$$\bullet \neg A \Rightarrow \neg C \tag{4.25}$$

and the safety goal for agent agA is:

$$\bullet\blacklozenge_{<T} D \Rightarrow \neg A \tag{4.26}$$

If a lockout agent agB is added to the system to prevent agent agA from violating the safety goal, the new shared indirect control relationship would be:

$$\bullet(A \wedge B) \Rightarrow C \tag{4.27}$$

$$\bullet(\neg A \vee \neg B) \Rightarrow \neg C \tag{4.28}$$

The safety goal for agents agA and agB would be:

$$\bullet\blacklozenge_{<T} D \Rightarrow \neg A \tag{4.29}$$

$$\bullet\blacklozenge_{<T} D \Rightarrow \neg B \tag{4.30}$$

4.5.2 Goal scope

Goal scope defines how closely the safety subgoals meet the system safety goal. It may be possible for agents to meet the original safety goal without restriction. However, it may sometimes be necessary or desirable to assign subgoals that are more restrictive than the original safety goal. This section identifies two categories of goal scope: *nonrestrictive* and *restrictive*.

Nonrestrictive. Nonrestrictive subgoals meet the parent goal with no additional limitations on functional behavior, other than what is defined in the parent goal. This is the base case where the system-level goal is fully realizable as defined. Subgoals *Achieve [StopBeforeHoistwayUpperLimit]* and *Achieve [EmergencyStopBeforeHoistwayUpperLimit]* from Section 4.5.1 are nonrestrictive if both the drive and emergency brake dynamics ensure the elevator will stop at the very end of the hoistway when the emergency brake is triggered, allowing full use of the hoistway.

Restrictive. In most systems, however, some amount of restriction is required to take into account uncertainty in system behaviors. A restrictive subgoal meets the parent goal but places additional limitations on system functionality. The most common restrictive subgoal is achieved with a *safety margin*, a hazard reduction technique for handling variability in failure rates of components [50]. Restrictive subgoals, which are usually less complex and easier to implement correctly and analyze than the parent goal, may be necessary if variables are not controllable, or if control delays are great.

In subgoals *Achieve [StopBeforeHoistwayUpperLimit]* and *Achieve [EmergencyStopBeforeHoistwayUpperLimit]*, *MaxStoppingDistance* and *MaxEmergencyBrakingDistance* could be increased so that the the elevator stops some distance before the hoistway limit, rather than at the end of the hoistway limit. The elevator would have less use of the full

hoistway, but the safety goal could be satisfied in the presence of jitter in *Drive* and *EmergencyBrake* actuation. In addition, the safety margin for the drive controller, the primary redundant subgoal, is usually designed to be even more restrictive than the safety margin for the emergency brake, to prevent the emergency brake from being activated frequently. If a safety goal has the form $\Box\ (A \leq B)$, then a subgoal with a safety margin C would be:

$$\Box(A \leq (B - C)) \tag{4.31}$$

In the goal elaboration of *Maintain[DoorClosedOrElevatorStopped]* shown in Table 4.3, the subgoals are more restrictive than the parent goal because they rely on worst-case actuation delays. Assume that *DoorController* and *DriveController* simultaneously issue commands to *DoorMotor* and *Drive* actuators respectively. If, in the following state, a controller sees that the combined commands could violate the goal, it can cancel its own actuation command before the actuation occurs. Technically, the safety goal would not have been violated until both actuation delays had finished, but the subgoals cancel the commands much earlier. This is an example of a type of safety margin used to restrict behavior.

OR reductions [66] are another common application of restriction. A goal of the form $\Box(A \vee B)$ is always satisfied if subgoal A is always satisfied. A is more restrictive than the parent goal because it excludes some functional behaviors that are non-hazardous: when A is false and B is true. OR reductions are common when one or more state variables in the parent goal are not observable or controllable by any agent in the system.

Restriction, though necessary, has its own limits (e.g., if the restrictions make the final product unusable). The goal*Maintain[ElevatorBelowHoistwayUpperLimit]* can always be met if the elevator is always stopped, but this trivial solution prevents functionality required in an elevator.

4.5.3 Controllability, observability, and alternative/restrictive goals

Choosing which variables to monitor and which to control to satisfy the parent goal depends on what sources of actuation and sensing are available, what software agent variables can be shared, and the general pattern of the goal. It may be possible to represent a particular goal with several different, but logically equivalent, goal patterns. The best representation is one that more-closely represents the intended observability and control relationships of the goal. For example, a goal pattern that uses a logical implication, such as $(A \Rightarrow B)$, is better for representing a goal that restricts the controlled variable B based on the observed variable A, whereas the logically equivalent goal pattern $\square(\neg A \vee B)$ is a better for representing a goal that restricts two variables controlled by the same agent.

Figure 2.5 lists all of the temporal logic operators that may be used the formal definition of the goals, subgoals, and indirect control relationships. However, some of these operators are used more often than others. In order for a goal to be realizable, its observed state variables must be observed at least one state prior to its controlled state variables.

In general it is most useful to define the goal such that controlled state variables are represented in the present state, and observed state variables are represented in some prior state. For example, the $\bullet$, $\bullet\blacksquare_{<T}$, and $\bullet\blacklozenge_{<T}$ operators are often used to represent values that are observed one state prior, for some duration T prior, and at least once in duration T prior to control of another state variable, respectively. Operators $\blacksquare$ and $\blacklozenge$ may also be used to represent observed variables, but are somewhat more difficult to operationalize because their duration of observation is unbounded. Variables that are intended to be controlled are usually represented in the present state, but may also be represented by operators $\circ$ and $\square$. Goals that contain $\lozenge$ are not realizable because they refer to some unbounded time in the future [46].

Table 4.5 shows the controllability and observability requirements for some common

Table 4.5. Goal controllability and observability requirements for realizability of goals of the form $A \Rightarrow B$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A ⇒ B	A		❑ ¬ A	Yes
	A	B	❑ ¬ A	Yes
	B		❑ B	Yes
	B	A	❑ B	Yes
	A, B			No
● A ⇒ B	A		❑ ¬ A	Yes
	A	B	❑ ¬ A	Yes
	B		❑ B	Yes
	B	A		No
	A, B			No
A ⇒ ● B	A		❑ ¬ A	Yes
	A	B	● ¬ B ⇒ ¬ A	No
	B		❑ B	Yes
	B	A	❑ B	Yes
	A, B		● ¬ B ⇒ ¬ A	No

entailment goals. When the goal takes the form $A \Rightarrow B$, both variables in the goal are in the same state. This means that both variables must be controllable by the same agent in order for that agent to satisfy the goal. This goal form is equivalent to $\neg A \vee B$. If only one variable is controllable, an OR reduction is formed on the goal to produce an alternate restrictive, but realizable goal.

When a goal takes the form $\bullet A \Rightarrow B$, then the goal is realizable without restriction if both A and B are controllable, or if A is observable and B is controllable. Otherwise, restricted goals are required.

When the goal takes the form $A \Rightarrow \bullet B$, then the goal is realizable without restriction if both A and B are controllable, or if B is observable and A is controllable. In the latter situation, the alternate goal $\neg \bullet B \Rightarrow \neg A$ is not restrictive. Rather, it is an equivalent

representation of the goal expression.

Common goal patterns and their controllability/observability requirements and alternative goals can be found in Appendix 7.2.

4.6 Conclusion

This chapter proposed the ICPA technique for safety goal elaboration. The concepts of direct and indirect control were defined with respect to the KAOS goal elaboration framework. In addition, the ICPA format and procedure were explained and illustrated with examples from a distributed elevator-control system. The chapter also contained a categorization of goal coverage strategies to use in safety goal elaboration, and an explanation of controllability and observability requirements for goal realizability, including goal patterns for those requirements and for alternative goals.

Chapter 5

Evaluation

5.1 Overview

In this Chapter, the ICPA technique is evaluated on a real semi-autonomous automotive system from a commercial automotive research lab. The system safety goals and the subgoals generated by applying ICPA are monitored at run-time in a simulation-based implementation of the automotive system in CarSim® and Simulink®. The results show that some system safety goal violations are detected in the subsystem monitors, but some are not, indicating that the subgoals produced with ICPA only partially compose the parent goals. In addition, monitoring at both the system and subsystem levels revealed defects in the system design and implementation and detected hazards that may be imperceptible to the driver.

5.1.1 Motivation

Evaluation of the ICPA technique should answer the following questions:

- Can ICPA be applied to a real system of non-trivial complexity?
- Do the subgoals represent a full or partial decomposition of the system safety goal?
- Are the subgoals produced by ICPA useful?

Any analysis technique related to system safety ultimately must be evaluated in a real complex safety critical system because the issues related to emergence are too easily abstracted away in a toy example. In a real system, it may not be possible to identify a full decomposition of a particular system safety goal. Guaranteeing that all emergence has been removed from the parent goal is not possible. The usefulness of the technique depends on how well it can be applied to a real safety critical system, with a believable level of complexity.

A working implementation is required, as well, for the same reason. The requirements and design specification are abstractions of the system that is actually built. The aim is for those abstractions to reflect the behavior of the true system. In reality, the built system may not conform to the requirements or design, or the requirements and design themselves may be incorrect. In order to evaluate the ICPA technique, it must be applied to a real system design, and the subgoals generated from it must be verified against a system implementation.

This chapter addresses the following question posed in Chapter 1:

- How do we evaluate the value of a partial decomposition?

5.1.2 Evaluation method

The method used to evaluate the ICPA technique in this thesis is proof of concept by case study. The application used in the case study was a semi-autonomous automotive vehicle. At the time the case study was performed, this automotive system was under development at a commercial automotive research lab. Requirements, design, and a simulation-based implementation of the automotive system were all partially complete. These materials were used to perform the following steps:

1. Define the system safety goals.

2. Apply ICPA to the system safety goals to define subgoals for the primary subsystems.

3. Add monitors of the safety goals and subgoals to the system implementation.

4. Monitor the safety goals and subgoals at run-time in a suite of driving scenarios.

Although defining the system safety goals is not officially part of the ICPA process, at the time of the study there were no requirements for the system under review that were specifically designated "system safety." Before ICPA could be applied, these safety goals were obtained by review of the functional requirements, design, and implementation.

Monitoring of the goals and subgoals during run-time scenarios was used to determine whether the subgoals represented a full or partial decomposition. In this thesis, a *hit* occurs when a goal violation is detected and a corresponding subgoal violation is detected. A *false positive* occurs when a subgoal violation is detected but no corresponding goal violation is detected. A *false negative* occurs when a goal violation is detected but no corresponding subgoal violations are detected. False negatives give an indication of the degree of emergence still remaining in the system. False positives either indicate the subgoals do not partially compose the parent goal, or redundant or restrictive goal coverage strategies are in use.

Run-time monitoring of safety goals and subgoals, in general, is useful for several purposes. Safety goal monitoring indicates whether or not the system is in a hazardous state. As noted before, a hazard implies that an accident could eventually occur, but does not necessarily indicate that an accident has occurred [50]. Subgoal monitoring indicates whether or not the subsystem is conforming to its safety subgoals. In combined monitoring, false positives can occur when subgoals are more restrictive than the parent goals, or if there remains some emergent behavior, Y in equation 3.23, that also satisfies the goal. False

negatives can occur when critical assumptions of the subgoals are violated, or when emergent behaviors remain in the system, X_i in equation 3.23, that prevent the subgoals from satisfying the goal.

5.2 Evaluation system

In Chapter 4, the various aspects of the ICPA technique were illustrated using a distributed elevator system from a graduate embedded systems course. This chapter presents the results from applying ICPA to a distributed embedded system under development at a commercial automotive research lab that is more complex than the distributed elevator example. This automotive system was being developed to study the use of advanced sensors and sensor fusion for a suite of semi-autonomous automotive features, not for a specific end-user commercial product. Although the implementation used for this thesis was fully simulated, subsequent implementations included hardware-in-the-loop and test vehicles.

5.2.1 Semi-autonomous automotive system

ICPA was applied to an automotive vehicle with safety-critical semi-autonomous features from an automotive research lab. At the time of this work, the system design, implementation, and simulation environment were functional but incomplete. The vehicle subsystems included in this study are shown in Figure 5.1. Other vehicle subsystems were excluded from this study.

The feature subsystems include two active safety features and three driver convenience features. Collision Avoidance (CA) detects objects in the forward path and stops the vehicle before a collision occurs. Rear Collision Avoidance (RCA) performs a similar function when the vehicle is moving in reverse, with the added behavior of detecting and stopping

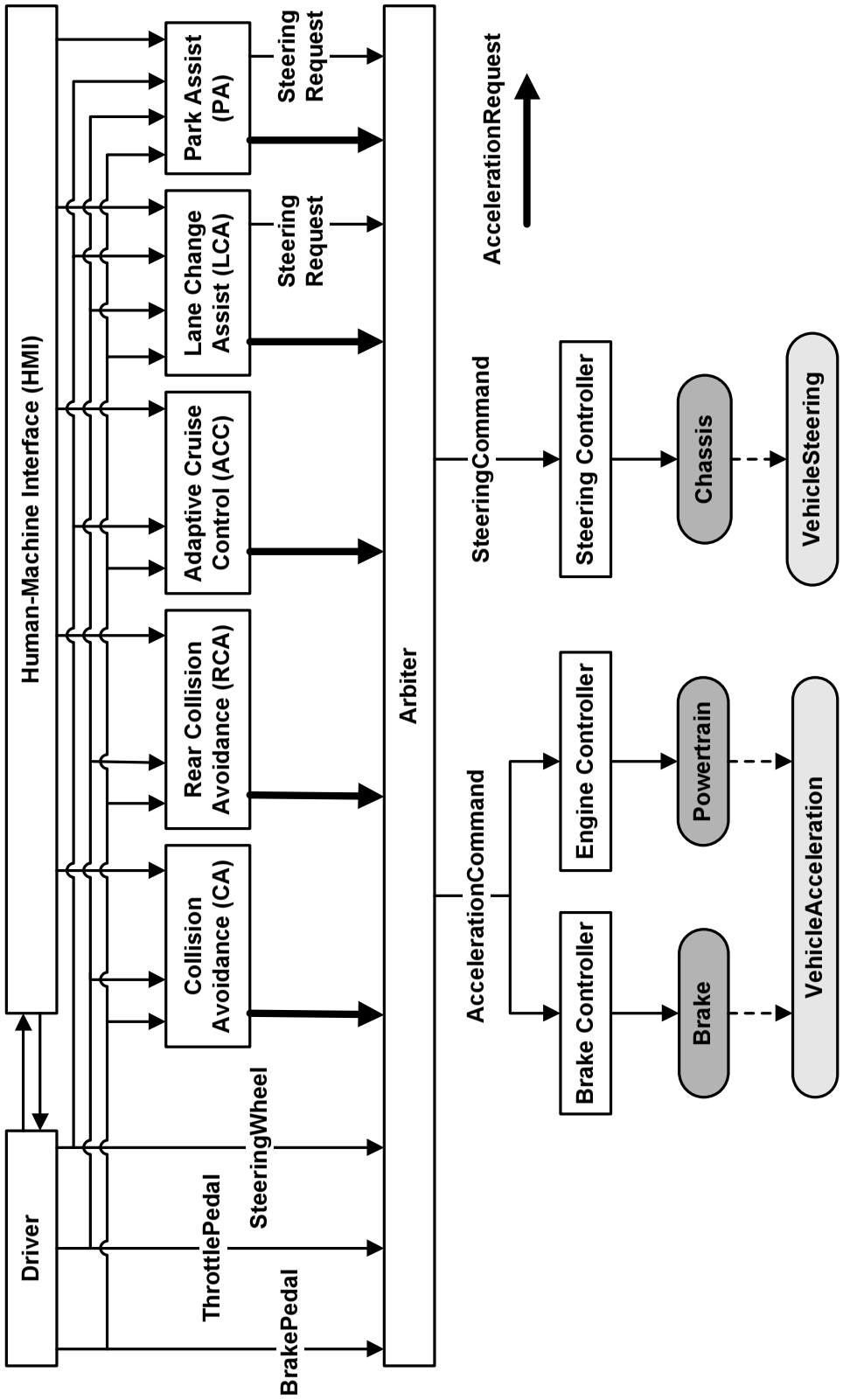


Figure 5.1. Semi-autonomous automotive system

for cross traffic behind the vehicle. Adaptive Cruise Control (ACC) commands the vehicle to a speed set by the driver, or to a set following distance behind a slower lead vehicle. Lane Change Assist (LCA) works in conjunction with ACC to perform a lane change maneuver when requested by the driver. Park Assist (PA) finds a parking space and parks the vehicle, also when requested by the driver.

In addition to the feature subsystems, the arbitration logic was included in this study as another subsystem along the indirect control path of the system safety goals. In the system implementation, this arbitration logic was distributed across multiple processors, with separate arbitration of acceleration and steering. For the purposes of this study, the Arbiter was treated as a single subsystem, rather than as a distributed subsystem. In other words, the behavioral decomposition, rather than the structural decomposition was used [41]. The behaviors of the feature subsystems were also distributed across a different structural decomposition in a similar manner.

This automotive system is semi-autonomous because subsystems control vehicle motion under driver supervision. The driver enables and disables the features, and is able to override feature control via the Human Machine Interface (HMI) or by application of the brake pedal, throttle pedal, or steering wheel. In CA and RCA, the driver has primary responsibility for detecting imminent collisions and avoiding them by some means, such as slowing, stopping, or steering the vehicle. In this case, CA and RCA provide backup safety behavior if the driver fails to stop the vehicle in time to avoid the collision. In ACC, LCA, and PA, the driver is responsible for monitoring the feature behaviors and intervening if needed (e.g., if a deer is heading toward the road but not yet detectable in the vehicle path). The driver may override any feature behavior except an emergency stop, in which case the driver is only permitted to brake harder than the feature.

5.2.2 Simulation platform

The vehicle system was implemented in the CarSim® simulation environment from Mechanical Simulation [64][1]. CarSim® is a software system for simulating vehicle dynamic responses to different acceleration and steering inputs. The software itself contains built-in models of vehicle systems and driving environments. For customized designs, models from other simulation environments such as Matlab® Simulink® [2] or Labview™ [3] can be added to the CarSim® models.

The system under review was implemented in a Simulink® model by the automotive research lab that designed it. Although some of the features were more fully implemented than others, no feature subsystem was complete. This model was then executed in CarSim®. The scenarios used in this thesis were performed on the vehicle model in CarSim® release 7.01 and Simulink® release r2008a running on Microsoft Windows XP.

5.2.3 Vehicle-level safety goals

In general, the system requirements were incomplete at the time of the study. In addition, none of the requirements for the research vehicle were written in a formal specification language and there were no requirements identified as system safety requirements. Thus, the first step that had to be completed before ICPA could be applied was to identify the system safety goals and define them formally. These safety goals were derived from inspection of the functional requirements, the hazard analysis that had been performed on the preliminary system requirements, and an incomplete system design and implementation. Tables 5.1 and 5.2 list the eight safety goals identified for this study.

The system safety goals define the safety-critical behaviors relevant to semi-autonomous vehicle motion. This includes both longitudinal acceleration and lateral steering. In this

Table 5.1. Safety goals for a semi-autonomous vehicle (1 of 2)

System Safety Goals for a Semi-Autonomous Automotive System (1 of 2)	
1.	**Goal:** Achieve[AutoAccelBelowThreshold] **InformalDef:** *Vehicle acceleration caused by autonomous vehicle control shall not exceed 2 m/s^2.* **FormalDef:** $\forall$ va: VehicleAcceleration IsSubsystem(va.source) $\Rightarrow$ va.value $\leq$ 2 m/s^2
2.	**Goal:** Achieve[AutoJerkBelowThreshold] **InformalDef:** *Vehicle jerk caused by autonomous vehicle control shall not exceed 2.5 m/s^3.* **FormalDef:** $\forall$ vj: VehicleJerk IsSubsystem(vj.source) $\Rightarrow$ vj.value $\leq$ 2.5 m/s^3
3.	**Goal:** Achieve[SubsystemAccelSteeringAgreement] **InformalDef:** *If a subsystem a) requests control of acceleration and steering and b) is granted control of either acceleration or steering, then the subsystem shall control both acceleration and steering.* **FormalDef:** $\forall$ va: VehicleAcceleration, vst: VehicleSteering, sn: SubsystemName ● RequestingAcceleration(sn) $\wedge$ ● RequestingSteering(sn) $\wedge$ ((va.source = sn) $\vee$ (vst.source = sn)) $\Rightarrow$ (va.source = vst.source = sn)
4.	**Goal:** Achieve[NoAutoAccelFromStop] **InformalDef:** *If a) the vehicle is stopped for a duration of StoppedTime and b) the throttle pedal has not been applied within the preceding GoTime and c) a subsystem is controlling acceleration and d) the HMI has not sent a go signal to the controlling subsystem within the preceeding AccelerationTime, then there shall be no vehicle acceleration* **FormalDef:** $\forall$ va: VehicleAcceleration, vsp: VehicleSpeed, sn: SubsystemName, hmi: HumanMachineInterface, tp: ThrottlePedal, st: StoppedTime, gt: GoTime (● ■$_{<st}$ IsStopped(vsp.value) $\wedge$ ● ■$_{<gt}$ $\neg$ @IsApplied(tp) $\wedge$ (va.source = sn) $\wedge$ ● ■$_{<gt}$ $\neg$ @Go(hmi, sn)) $\Rightarrow$ $\neg$ IsAccelerating(va.value)

Table 5.2. Safety goals for a semi-autonomous vehicle (2 of 2)

System Safety Goals for a Semi-Autonomous Automotive System (2 of 2)	
5.	**Goal:** Achieve[DriverForwardAccelOverride] **InformalDef:** *If a) the vehicle is moving in the forward direction and b) the driver is applying the brake pedal or the throttle pedal and c) a subsystem is requesting a vehicle acceleration greater than or equal to -2 m/s^2 (i.e., not requesting a "hard" stop of the vehicle), then the subsystem shall not control vehicle acceleration.* **FormalDef:** $\forall$ va: VehicleAcceleration, sn: SubsystemName, bp: BrakePedal, tp: ThrottlePedal $\bullet\bullet$ (InForwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active) $\wedge$ RequestingAcceleration(sn) $\wedge$ (RequestedAcceleration(sn) $\geq$ -2 m/s^2)) $\Rightarrow \neg$ (va.source = sn)
6.	**Goal:** Achieve[DriverBackwardAccelOverride] **InformalDef:** *If a) the vehicle is moving in the backward direction and b) the driver is applying the brake pedal or the throttle pedal and c) a subsystem is requesting a vehicle acceleration less than or equal to 2 m/s^2 (i.e., not requesting a "hard" stop of the vehicle), then the subsystem shall not control vehicle acceleration.* **FormalDef:** $\forall$ va: VehicleAcceleration, sn: SubsystemName, bp: BrakePedal, tp: ThrottlePedal $\bullet\bullet$ (InBackwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active) $\wedge$ RequestingAcceleration(sn) $\wedge$ (RequestedAcceleration(sn) $\leq$ 2 m/s^2)) $\Rightarrow \neg$ (va.source = sn)
7.	**Goal:** Achieve[DriverSteeringOverride] **InformalDef:** *If the driver is turning the steering wheel, then no subsystem shall control vehicle steering* **FormalDef:** $\forall$ vst: VehicleSteering, sn: SubsystemName, sw: SteeringWheel $\bullet\bullet$ (sw.active) $\Rightarrow \neg$ (vst.source = sn)
8.	**Goal:** Achieve[ForwardBlockAccelSteering] **InformalDef:** *If the vehicle is moving forward, then the subsystem RCA shall not control vehicle acceleration or steering* **FormalDef:** $\forall$ va: VehicleAcceleration, vst: VehicleSteering, vsp: VehicleSpeed $\bullet\bullet$ InForwardMotion(vsp.value) $\Rightarrow \neg$ ((va.source = `RCA') $\vee$ (vst.source = `RCA'))
9.	**Goal:** Achieve[BackwardBlockAccelSteering] **InformalDef:** *If the vehicle is moving backward, then the subsystems CA, ACC, and LCA shall not control vehicle acceleration or steering.* **FormalDef:** $\forall$ va: VehicleAcceleration, vst: VehicleSteering, vsp: VehicleSpeed $\bullet\bullet$ InBackwardMotion(vsp.value) $\Rightarrow \neg$ ((va.source $\in$ {CA, ACC, LCA}) $\vee$ (vst.source $\in$ {CA, ACC, LCA}))

system, all features attempt to control acceleration, and LCA and PA attempt to control steering. CA, ACC, and LCA are only operational when the vehicle is in forward motion, RCA is only operational in reverse, and PA operates in both.

In semi-autonomous automotive systems, the driver remains engaged in the driving process, even when subsystem features are controlling the vehicle. Autonomous control is both initiated and overridden by the driver. As a result, many of the safety goals for semi-autonomous driving constrain behaviors that interfere with the ability of the driver to properly supervise the vehicle. Autonomous behaviors must be performed in a way that allows the driver enough time to identify what is happening and intervene if necessary. One historic concern for general automotive systems is unintended, or sudden acceleration [71] [77]. Although most incidents of unintended acceleration have been attributed to incorrect pedal application (hitting the throttle instead of the brake pedal), as vehicles move closer to fully autonomous control, it becomes increasingly possible for such accelerations to be caused by system defects.

The first guard against unintended accelerations is safety goal number 1, which prohibits autonomous vehicle acceleration that exceeds 2 m/s^2. The second is safety goal number 2, which prohibits autonomous vehicle jerk that exceeds 2.5 m/s^3. These are reasonable limits of comfortable acceleration and jerk in vehicle motion [58]. Note that the vehicle is allowed to cause uncomfortable levels of negative acceleration (deceleration) because the vehicle may need to perform a sudden, uncomfortable stop to avoid a collision. Features that increase vehicle speed, such as ACC, LCA, or PA, are limited to more gradual accelerations, to give the driver time to override if necessary.

Another guard against unintended accelerations is safety goal number number 4, which prohibits autonomous accelerations when the vehicle is stopped (e.g., if ACC brings the vehicle to a stop behind a stopping lead vehicle or if CA stops the vehicle to avoid an object

in the road). In these situations the driver is required to initiate acceleration by sending a "Go" signal from the Human-Machine Interface (HMI), or by pressing the throttle pedal. This ensures that the driver is aware the vehicle will begin moving from a stopped position.

Another concern is feature interaction [13], which was discussed in more detail in Section 2.4.1. A feature interaction occurs when the combined behavior of two features is undefined in the specification of either and must be different than the linear superposition of the individual features. In the semi-autonomous automotive system under review, several different features may attempt to control vehicle acceleration or steering, all at the same time. If one feature controls steering while another controls acceleration, a behavior may occur that is undefined by either, or by the system specification. Safety goal number 3 prohibits control of acceleration and steering by different subsystems if either is attempting to control both.

Safety goals 5, 6, and 7 ensure the driver is allowed to override acceleration and steering. Driver brake and throttle commands override acceleration control, unless the subsystem feature is attempting a very hard brake of the vehicle. Because prior analysis of sudden acceleration incidents has shown that drivers sometimes hit the accelerator pedal instead of the brake pedal [71] [77], the features used for collision avoidance are allowed to override driver throttle or brake pedal commands.

Finally, safety goals 7 and 8 prohibit features from controlling vehicle acceleration or steering when the vehicle is moving in a direction in which the features were not intended to operate. These goals essentially define which subsystems are allowed to perform vehicle control actions in both the forward and reverse directions.

5.3 ICPA and subsystem subgoals

Once the system safety goals were determined, ICPA was applied to identify subgoals for the subsystems. Appendix C contains the ICPA for all nine goals listed in Tables 5.1 and 5.1.

The goal coverage strategy contains a redundant responsibility goal assignment for eight of the nine goals (1-2 and 4-9) As the final source of system acceleration and steering commands, the Arbiter becomes the primary source for meeting the system safety goal. The secondary safety goals of the feature subsystems provide protection against some single-point failures of the Arbiter. If the Arbiter fails by choosing an acceleration or steering command from the wrong feature subsystem source, then the system should still meet the safety goal. However, if the Arbiter fails in a different way (e.g., summing requests from different features), then the features will not provide backup protection against the failure.

The goal coverage strategy for goal 3 is single responsibility, with the Arbiter as the only source satisfying the goal. This goal prohibits behaviors that mix acceleration and steering control from different feature subsystems. An alternative redundant responsibility strategy could require one feature subsystem to monitor the other feature subsystem control requests and cancel their requests when higher-priority features request control. However, this requires the arbitration logic to be maintained in each feature subsystem, which is impractical in this distributed development environment.

The goal scope for all the goals is restrictive in some way or another. In all goals, worst-case actuation delays are used for defining the subgoals. In some goals, OR-reduction is used to limit the behavior of the feature subsystems. For example, goals 1 and 2 use OR-reduction. The subgoals for the feature subsystems limit the value of acceleration requests when those acceleration requests are the source of vehicle acceleration. However, it is simpler to always prohibit the subsystems from requesting excessive vehicle acceleration

or jerk, rather than prohibiting it only when those requests are used to control vehicle acceleration.

5.3.1 Goal and subgoal monitoring locations

The monitoring locations for the goals and subgoals are presented in Table 5.3. For goals 1, 2, and 4, the actual goal that is monitored at the system level is different from the goal that is monitored for the Arbiter. For goals 3 and 5-8, the actual goal that is monitored at the system level is the same as the goal that is monitored at the Arbiter level. This is because the goals constrain state variables that have some source of direct control in the subsystem. For example, there is no way to sense which subsystem is the source of vehicle acceleration. The only way to monitor this is to monitor the *source* tag on the acceleration command. However, vehicle acceleration is not directly controlled by any subsystem, and can be monitored by an accelerometer. A goal can be monitored at the system level if it constrains one or more sensed state variables or two or more state variables directly controlled by different subsystems.

5.3.2 Lessons from applying ICPA

Applying the ICPA revealed the following information about the semi-autonomous automotive system:

- Arbitration of feature subsystem control requests is divided between longitudinal acceleration and steering. This complicates actions that coordinate the two types of vehicle control.

- Prioritization of feature subsystem control requests in steering arbitration is the reverse of the prioritization in the acceleration arbitration. This can lead to feature

Table 5.3. Monitoring locations of goals and subgoals.

Goal/Subgoal		Monitored In:						
		Vehicle	Arbiter	CA	RCA	ACC	LCA	PA
1	**Achieve[AutoAccelBelowThreshold]**	X						
1A	Achieve[AutoAccelCommandBelowThreshold]		X					
1B	Maintain[AutoAccelRequestBelowThreshold]			X	X	X	X	X
2	**Achieve[AutoJerkBelowThreshold]**	X						
2A	Achieve[AutoJerkCommandBelowThreshold]		X					
2B	Maintain[AutoJerkRequestBelowThreshold]			X	X	X	X	X
3	**Achieve[SubsystemAccelSteeringAgreement]**							
3A	Achieve[SubsystemAccelSteeringCommandAgreement]		X					
4	**Achieve[NoAutoAccelFromStop]**	X						
4A	Achieve[NoAutoAccelCommandFromStop]		X					
4B	Achieve[NoAutoAccelRequestFromStop]			X	X	X	X	X
5	**Achieve[DriverForwardAccelOverride]**							
5A	Achieve[DriverForwardAccelOverrideAccelCommand]		X					
5B	Achieve[DriverForwardAccelOverrideAccelRequest]			X	X	X	X	X
6	**Achieve[DriverBackwardAccelOverride]**							
6A	Achieve[DriverBackwardAccelOverrideAccelCommand]		X					
6B	Achieve[DriverBackwardAccelOverrideAccelRequest]			X	X	X	X	X
7	**Achieve[DriverSteeringOverride]**							
7A	Achieve[DriverSteeringOverrideSteeringCommand]		X					
7B	Achieve[DriverSteeringOverrideSteeringRequest]						X	X
8	**Achieve[ForwardBlockAccelSteering]**							
8A	Achieve[ForwardBlockAccelSteeringCommand]		X					
8B	Achieve[ForwardBlockAccelSteeringRequest]				X			
9	**Achieve[BackwardBlockAccelSteering]**							
9A	Achieve[BackwardBlockAccelSteeringCommand]		X					
9B	Achieve[BackwardBlockAccelSteeringRequest]			X		X	X	

interaction problems if different feature subsystems are chosen to control acceleration and steering.

- The Arbiter indicates if a feature subsystem or the driver has control over longitudinal acceleration with separate 'selected' flags. This could allow control actions to be attributed to multiple sources, making it difficult to identify the true source of vehicle control.

- ACC performs the longitudinal control for LCA. Thus ACC and LCA share acceleration requests. Monitoring of subgoals that limit acceleration requests is not necessary for LCA, if the same subgoals are monitored for ACC.

Applying the ICPA revealed the following information about the ICPA process itself, as well as the subgoals it produced:

- Almost all safety subgoals are restrictive, often in multiple ways. Most goal restriction comes from variability, or jitter, in the values monitored or controlled by the system agents.

- Some goals can only be monitored at the subsystem level. If the goal restricts control of a state variable directly controlled by some agent in the system, then the highest level in the system hierarchy at which the goal can be monitored is the level containing the subsystem that controls the variable. If the goal restricts two or more state variables directly controlled by different agents in the system, or one or more sensed from system dynamics or the environment, then the goal can be monitored separately from the subgoals.

- Goal redundancy between levels in a system's behavioral hierarchy can only protect against defects that occur in subsystems located earlier in the control flow. It does

not protect against defects in the subsystems located later in the control flow. If goal redundancy is not used at the latest stage in control flow (i.e., the direct control source or indirect control source closest to it), then goal redundancy does not protect against single-point failures.

5.4 Evaluation scenarios and results

Although it is impossible to determine if the unknown/unrealizable part of the system safety goal has been minimized, it is possible to determine whether or not the subgoals produced by ICPA partially compose the parent goal. The purposes of goal elaboration are twofold. First, it is important to identify subsystem behaviors that contribute to hazards at design-time so that the subsystem development team can design and build the subsystem to help satisfy the system goal. A second aim is to monitor the subsystem behaviors at run-time to determine whether a subsystem is violating its safety goals. Violations of subgoals may be predictors of system-level goal violations. In this thesis, the safety goals and subgoals were monitored in an implementation of a vehicle in a simulation environment.

At the time this evaluation was performed, although all features were partially functional, no feature was fully complete. In addition, the features were implemented independently of the process for defining and elaborating the system safety goals. That is, the formal specification of the safety goals and subgoals was not available to the teams designing and implementing the system. The safety goals were obtained by the author of this thesis by analysis of the functional requirements, hazard analysis, and partial implementation. As such, this work cannot determine whether applying ICPA to obtain subgoals helps in the design and implementation of the system. Rather, the purpose of this analysis is to determine whether system safety in this case is partially, if not fully, composed by the set of subgoals obtained from ICPA. To do this, we monitored the system safety goals

and subgoals at run-time in ten different driving scenarios, configured in CarSim® and Simulink®.

The following ten scenarios were chosen to test the goal and subgoal monitors. They are representative of real driver behaviors, both those that the driver is expected to do regularly, such as engage ACC at a reasonable speed, and those that the driver might do in error, such as engage PA in the middle of an emergency braking action. Each scenario was scheduled for a simulation time of 20 s. The goal and subgoal violations for all ten scenarios are listed in Tables D.1-D.11 in Appendix 7.2. Descriptions of each scenario and evaluations of their results are presented below.

5.4.1 Scenario 1: CA enabled, ACC enabled, stopped vehicle in path

Description. The host vehicle is traveling forward, starting from a stop 20 m behind another stopped vehicle. ACC is enabled ("turned on" by the driver), but not engaged (operating in an internal state in which vehicle control actions are performed). CA is enabled. CA is expected to initiate a hard braking action to stop the host vehicle before a collision occurs.

Results. Table D.1 lists the goal violations for the first scenario. CA attempted to stop the host vehicle upon its approach to the stopped vehicle in its path. This resulted in violation of vehicle safety goals 1 and 2 shortly before early termination of the simulation, at simulation time 12.681 s. The longest violation was 8 ms, and the shortest was 1 ms (the time interval of one state). Vehicle acceleration exceeded its threshold for 4 ms, at time 12.589 s, 92 ms before early termination. Although the vehicle acceleration threshold was exceeded once, no violations of the corresponding subgoals were also detected.

Vehicle jerk was exceeded six times, for 8, 2, 1, 4, 6, and 1 ms, at 98, 83, 29, 20, 9, and

1 ms before early termination of the system. Similarly, although the vehicle jerk threshold was exceeded six times, only violations in corresponding subgoals for CA and PA were detected. The CA jerk threshold was violated only once for 1 ms, starting 80 ms before early termination of the system. In addition, the jerk threshold for PA was violated once, once for 1 ms at time 0.001 s, and once for 1ms at time 9.624 s, 3.057 s before early termination of the system.

These results indicate that the subgoals do not fully compose the parent goal. Although there was one violation of goal 1, there were no corresponding violations of subgoals 1A and 1B. In addition, there were six violations of goal 2, but no violations of subgoal 2A, and only three violations of 2B.

The results also indicate that there is restriction or redundancy built into the subgoals. The subgoal 4B, no autonomous acceleration from a stop, was violated for PA at the start of simulation time, but no violations were detected in 4 or 4A. The violations of 2B for park assist appear to be false negatives; they do not appear to correspond to any of the violations of 2.

A closer look at the acceleration requests for CA and PA illustrate what is happening in the vehicle. Figure 5.2 shows the value of CA's acceleration request at the end of simulation time. It indicates that CA requests a hard brake of the system, but immediately releases it. This violates the jerk goal for CA, but only for one state. The jerk goal for the Arbiter was not violated, because when CA released the brake, control was also transferred back to the driver. To the Arbiter, the jerk was violated by the driver's throttle and brake pedal inputs that took over when CA released the brake, not CA releasing the brake itself.

In actuality, a step-increase in the acceleration request alone should not violate the system level safety goal because the vehicle response dynamics (e.g., inertia of the physical mass of the vehicle) would make the transition from the current acceleration to the newly

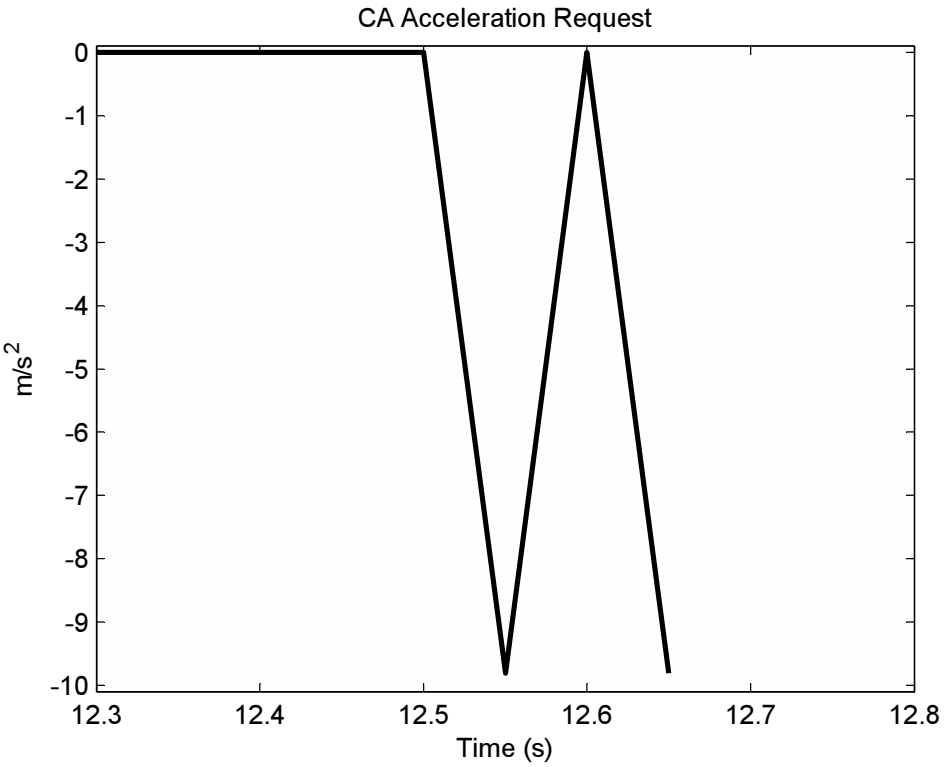


Figure 5.2. Scenario 1: CA begins a braking action, but cancels it briefly before beginning it again.

requested acceleration more gradual. In this case, the subgoal is perhaps too restrictive for what is practical to design. However, the sudden release of the hard stop, is incorrect CA behavior. CA should continue braking until the vehicle stops, and then hold the vehicle at that position until the driver initiates motion by applying the throttle pedal. It is possible that the vehicle dynamics, in response to this sudden brake application and release, behaved in a way that caused the other vehicle safety goals to be violated (the acceleration threshold and jerk thresholds), because all of those violations occurred after this action by CA.

Further inspection of the PA acceleration requests, shown in Figure 5.3, also reveals incorrect behavior by the PA. In this scenario, PA requests an acceleration of 2 m/s^2 from the start of simulation time to time 2.186 s, at which time PA requests no acceleration. At

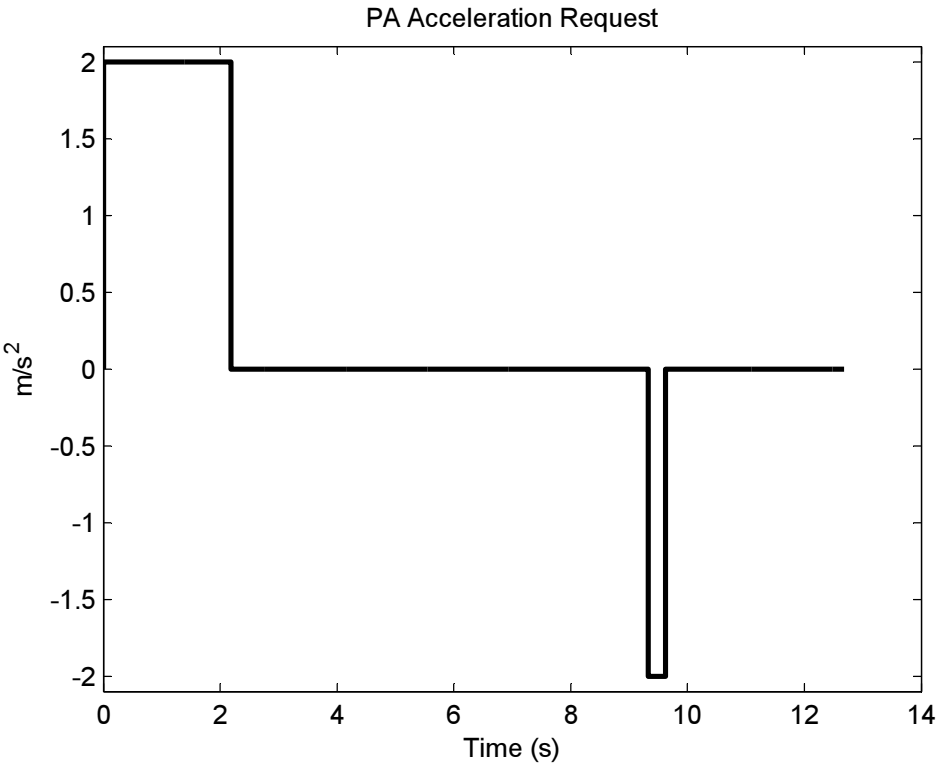


Figure 5.3. Scenario 1: PA requests acceleration without being enabled.

time 9.33 s, PA then requests an acceleration of -2 m/s^2 until 9.624 s, when it again requests an acceleration of 0 m/s^2. Subgoal 4B was violated by this behavior because at the start of simulation time the vehicle is stopped. The jerk thresholds were also violated each time the acceleration requests switched from low to high. PA was never selected by the Arbiter to control vehicle acceleration because PA never signaled that it was active. However, this behavior is incorrect and potentially unsafe if a fault occurs that flips the PA active signal, or if the Arbiter passes along the acceleration request without looking at the PA active flag.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- PA is sending out acceleration requests when it is not enabled.

- Goal redundancy in the Arbiter blocks the improper acceleration requests from PA.
- The vehicle exceeds the acceleration limit for autonomous control shortly before the simulation terminates in error. The two behaviors may be related to the same underlying design or implementation defect.

Run-time monitoring in this scenario also revealed the following additional information about the subgoals and ICPA:

- The jerk threshold goal is too restrictive to be implemented practically. A revised goal would allow jerk thresholds to be violated for a single state.
- Goal 1 is not fully composed by subgoals 1A and 1B. This may indicate some of the indirect control relationships used in the ICPA are incorrect.

5.4.2 Scenario 2: CA engaged, ACC enabled, PA enabled, stopped vehicle in path

Description. The host vehicle is traveling forward, starting from a stop 20 m behind another stopped vehicle. ACC is enabled, but not engaged. CA is enabled. Just after CA begins to perform a hard braking action at time 12.55 s to avoid the stopped vehicle, the driver engages PA at time 12.56 s. CA is expected to remain in control of vehicle acceleration and stop the host vehicle.

This scenario was chosen to verify a design defect in the arbitration logic found while reviewing the system requirements and design for the ICPA. In the design, acceleration and steering were arbitrated separately. Inspection during ICPA indicated that prioritization of feature requests in steering arbitration was reversed. The vehicle should continue to stop and remain stopped, despite application of PA

Results. Table D.2 lists the goal violations for Scenario 2. As in Scenario 1, CA attempted to stop the host vehicle upon its approach to the stopped vehicle in its path. Scenario 2, however, resulted in violation of vehicle safety goals 1-3, also before early termination of the simulation, this time at simulation time 12.588 s. In this situation, vehicle acceleration was exceeded for 1 ms at time 12.587 s, vehicle jerk was exceeded for 20 ms at time 12.561 s, and the acceleration-steering agreement goal was violated for 27 ms at time 12.561 s. Each vehicle safety goal was still in violation at the time of early termination, which occurred 93 ms earlier than early termination in the first scenario.

Similar to the first scenario, no subgoals for goal 1 were violated. In addition, subgoal 2A for the Arbiter was violated only once for 1 ms at time 12.561 s, 7 ms before vehicle jerk was exceeded and 27 ms before early termination of the system. As in the first scenario, the jerk subgoal 2B for PA was violated twice, once at the start of simulation and once at time 9.624 s.

The acceleration/steering agreement subgoal 3A for the Arbiter was the same as the system-level goal, as noted in Section 5.3.1, thus violations of the Arbiter's goal corresponded to those of the system.

Plots of the arbiter's acceleration command, CA's acceleration request, and CA's 'selected' tag are shown in Figure 5.4. CA was selected as the source of the acceleration command at time 12.551 s, and remained selected until the simulation terminated. The acceleration command was set to CA's requested acceleration at time 12.551 s, but was reset to 0 m/s^2 at time 12.561, 1 ms after PA was enabled (one system state later). Thus, the Arbiter was indicating that CA was chosen to be the source of the acceleration command, but was choosing PA's acceleration request as the actual source.

This confirmed the design defect in the arbitration logic discovered during the ICPA. In the design, arbitration was divided by acceleration and steering, with acceleration arbitra-

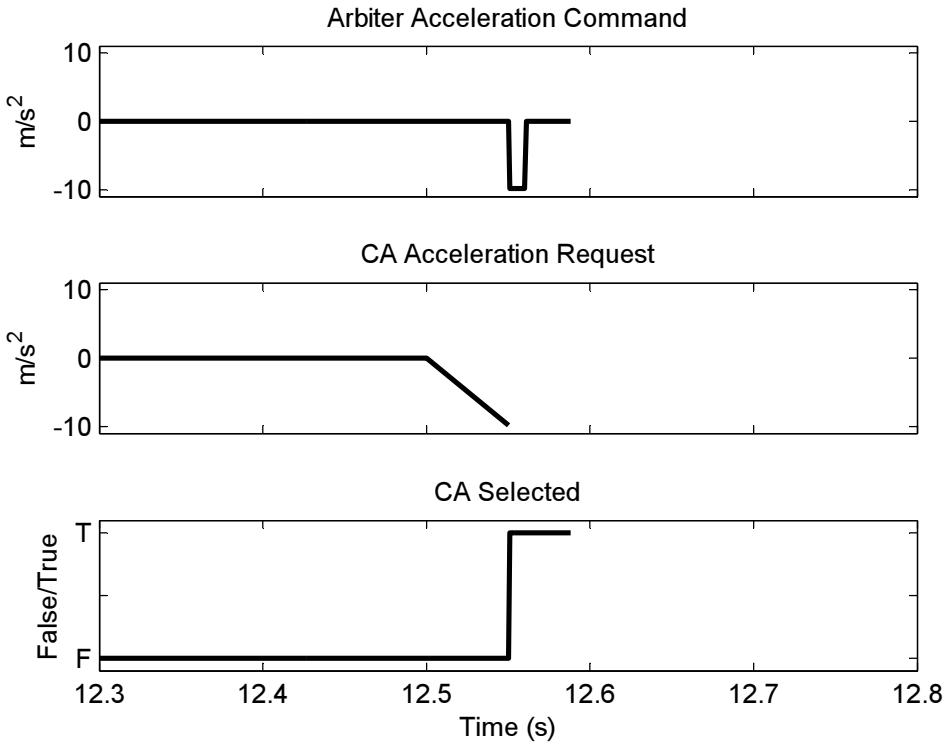


Figure 5.4. Scenario 2: CA is not the source of the acceleration command when PA is enabled, even though CA is selected to be in control of acceleration.

tion occurring first. In the steering component of arbitration, the prioritization order was reversed. Although arbitration of acceleration determined how the CA_Selected tag was set, arbitration of steering actually determined what values of both acceleration requests and steering requests were passed along as acceleration commands. Thus, the Arbiter was selecting CA as the source, but sending out PA's requests.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- The reverse arbitration logic found during application of the ICPA was confirmed.
- The Arbiter is a potential source of single-point failures of the system.

5.4.3 Scenario 3: CA engaged, ACC enabled, throttle pedal applied, stopped vehicle in path

Description. The host vehicle is traveling forward, starting from a stop 20 m behind another stopped vehicle. ACC is enabled, but not engaged. CA is enabled. Just after CA begins to perform an emergency braking action at time 12.55 s to avoid the stopped vehicle, the driver applies the throttle pedal at time 12.56 s. CA is expected to remain in control of vehicle acceleration and stop the host vehicle.

Results. Table D.3 lists the goal violations for the third scenario. In this scenario, CA began to stop the host vehicle, but canceled its braking request, allowing the vehicle to 'hit' the parked vehicle, and the simulation terminated normally at 20s. (CarSim® does not simulate vehicle collisions, therefore, in the simulation the host vehicle passed through the stopped vehicle in its path).

Goal 1, the acceleration threshold, was violated once at time 13.652 s for a duration of 3 ms. Goal 2, the jerk threshold, was violated three times, at times 12.565 s for 2 ms, at time 12.902 s for 3 ms, and at time 12.913 s for 4 ms. Subgoal 2A was not violated. Subgoal 2B was violated four times for CA, from time 12.6 s to time 13.85 s, each for a duration of 1 ms. Subgoal 2B was violated 47 times by ACC/LCA, for a duration of 1 ms each, starting at time 12.75 s and ending at time 15.6 s.

PA experienced the same violations of goals 2B and 4B as in the first two scenarios. Goal 5, which allows the driver to override acceleration if the subsystem is not performing a hard brake, was violated four times between times 12.562 s, with durations ranging from 4 ms to 61 ms. Its subgoal, 5B was violated five times by CA, between times 12.6 s and 13.85 s, with durations ranging from 50 to 200 ms. PA experienced the same violations of goals 2B and 4B as in the first two scenarios.

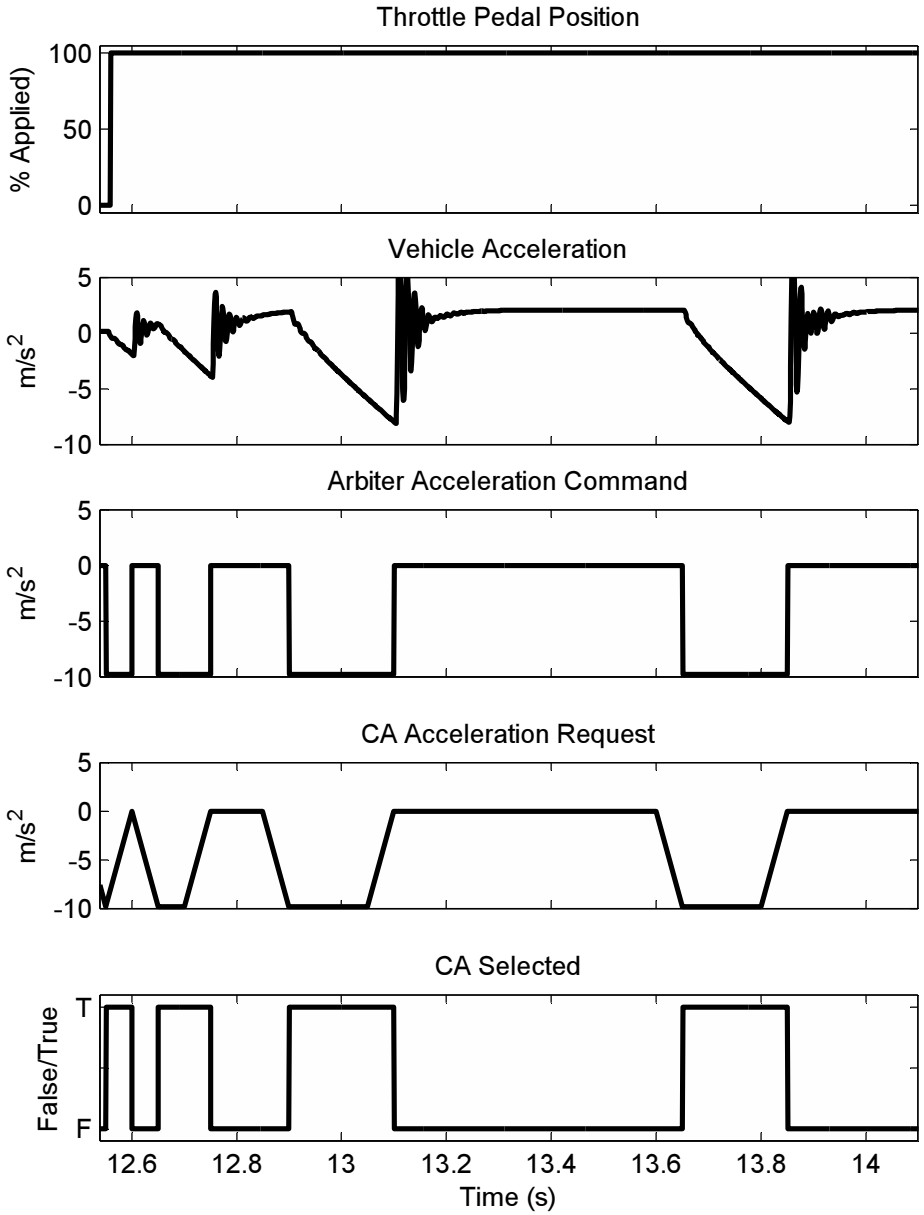


Figure 5.5. Scenario 3: CA engages to stop the host vehicle, even though throttle pedal is applied. The CA braking action is intermittent, however, and fails to stop the host vehicle before 'hitting' the parked vehicle in its path.

Plots of the throttle pedal position, vehicle acceleration, acceleration command, CA acceleration request, and CA selected variables are shown in Figure 5.5. Each time CA applies the brake, it is selected by the Arbiter as the source of the acceleration command. However, its request of -10 m/s^2 is not achieved by the vehicle because acceleration is delayed by the vehicle response. During the time that the vehicle acceleration is not below -2 m/s^2, safety goal 5 is violated. This corresponds to the four times CA is actively requesting a hard brake. The Arbiter's subgoal 5A is not violated because the Arbiter never sends acceleration commands for CA greater than -2 m/s^2. Subgoal 5B is violated five times by CA. This occurs in-between the hard brake requests, when CA is engaged but entering a preparation state in which no deceleration is requested. In these situations, CA is not selected as the winner of arbitration over the driver, even though CA is engaged, because CA's requested acceleration is not less than -2 m/s^2.

Plots of the throttle pedal position, vehicle speed, acceleration command, ACC acceleration request, and ACC selected are shown in Figure 5.6. The data show that ACC is sending out requests to control the vehicle to a set speed of 0 m/s, even though ACC has only been enabled, not engaged. When the CA braking action completes, ACC decreases this deceleration request gradually back to zero acceleration. During this time, the vehicle is stopped. Once the vehicle has been stopped for some time, ACC should wait for another throttle application (i.e., a switch from not applied to applied) or a go signal from the HMI.

The results from this scenario show that both PA and ACC are sending acceleration requests when they are not engaged. Although they both have 'active' tags to indicate whether or not these requests are intentional, it would be better to not request acceleration when it is not really needed. The results also show that the vehicle correctly continues to perform the emergency stop when CA requests it, even though the driver is pressing the throttle pedal. However, the braking actions performed by CA are insufficient to stop the

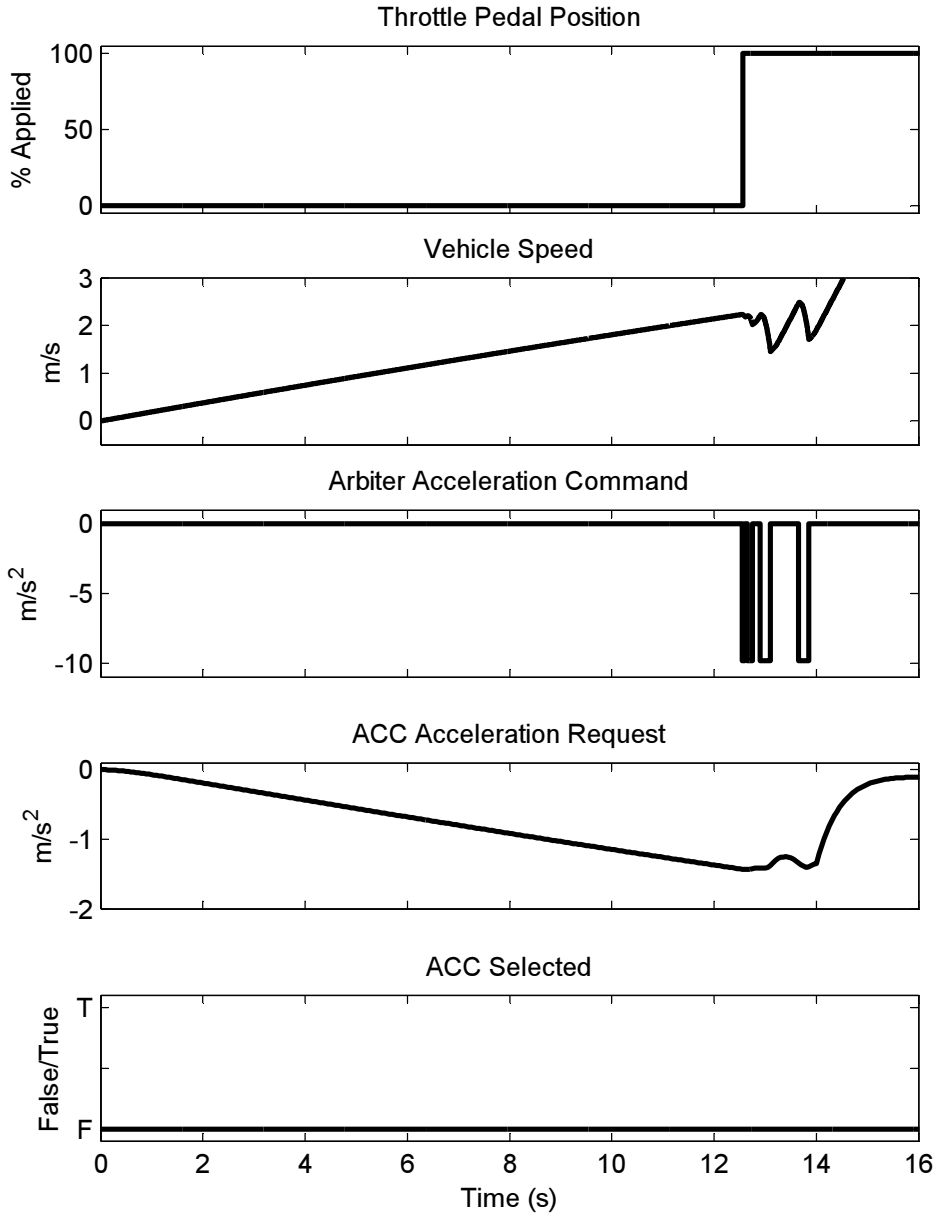


Figure 5.6. Scenario 3: ACC sends acceleration requests to control the vehicle to a set speed of 0 m/s, even though ACC is not engaged.

vehicle.

The only difference between this scenario and the first is the application of the throttle pedal. It is possible that CA begins the same type of intermittent braking action in both scenarios, but in this scenario the added application of the throttle pedal allows the simulation to continue. In this case, perhaps the intermittent braking action alone causes instability in the vehicle dynamics. It is also possible that the application of the throttle pedal causes the intermittent nature of the braking action. In either situation, a closer examination of CA is required to troubleshoot the problem.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- CA appears to continue to execute a hard brake when the driver applies the throttle pedal. This is the intended behavior of the system.
- CA appears to execute the hard brake in a manner that is intermittent, rather than continuous. This may be insufficient for braking the vehicle in time to avoid the collision.

Run-time monitoring in this scenario also revealed the following additional information about the goals obtained from ICPA:

- The nature of when CA performs its braking actions was misunderstood during the ICPA. There are some states in which CA is engaged, but not performing a braking action. These states should be excluded from the analysis of when CA is active.
- The indirect control relationships between the CA acceleration request, Arbiter acceleration command, and resulting vehicle acceleration do not perform as defined in

the ICPA. There is a greater delay between the acceleration command and the resulting acceleration. System goal 5 should be adjusted to account for these delays. Subgoals 5A and 5B, however, seem to be correct. In this case, monitoring revealed a parent goal that is too restrictive to be satisfied while still satisfying the functional behavior of the CA feature subsystem.

5.4.4 Scenario 4: throttle pedal applied, ACC engaged, CA enabled, slow vehicle in path

Description. The host vehicle is traveling forward, starting 25m behind another vehicle that is traveling between 10-25 k/h (2.78-6.94 m/s). The driver accelerates the host vehicle by applying the throttle pedal from time 0.5 s to 8.5 s. At time 2.0 s, ACC is engaged by the driver. As the host vehicle approaches a slower lead vehicle, ACC is expected to slow the host vehicle to follow a set distance behind the slower lead vehicle.

Results. Table D.4 lists the goal violations for the fourth scenario. In this scenario, ACC attempted to slow down the host vehicle when it approached the slower lead vehicle, but the simulation terminated early at time 13.142 s.

Goal 1, the vehicle acceleration threshold, was violated ten times between time 12.987 s and time 13.127 s. The longest violation was 67 ms, and the shortest violation was 1 ms. No subgoals for this goal were violated for the Arbiter or features. Goal 2 was violated 98 times between time 8.617 s and time 9.529 s. Its subgoal 2A was violated 48 times between time 9.051 s and 13.051 s. Subgoal 2B was violated 49 times by ACC/LCA, from time 2.05 s to time 13.05 s, and twice by PA at times 0.001 s and 3.906 s. Once again, PA violated subgoal 4B, but this time only at the start of the simulation. Goal 5, which allows the driver to override all but hard brake requests, was also violated at time 2.052 s.

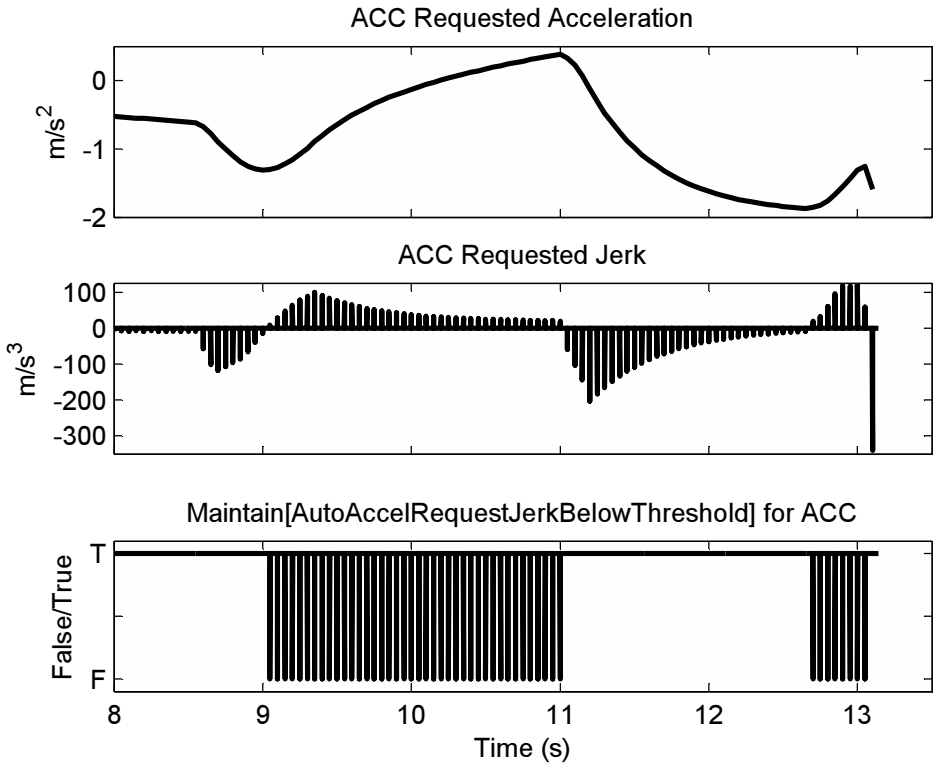


Figure 5.7. Scenario 4: ACC acceleration request and jerk profile.

Its subgoals 5A and 5B for the Arbiter and ACC were also violated, at times 2.051 s and 2.0 s, respectively. Finally, the two goals that block certain subsystems from controlling the vehicle in forward and backward motion, goals 8 and 9 respectively, were each violated once for 1 ms. Goal 9 was violated at time 13.074 s, withs subgoals 9A for the Arbiter and 9B for ACC were also violated at time 13.074 s.

Figure 5.7 shows ACC's requested acceleration, the resulting jerk of that acceleration, and the jerk threshold goal for ACC. All jerk violations were 1 ms violations. That is, they occurred from a change in acceleration that occurred in a single state. Although it may be possible to design out most of these jerk violations, it is probably not practical to do so. Rather, this seems to indicate that the subgoal is too restrictive for normal vehicle use.

Figure 5.8 shows throttle pedal position, vehicle acceleration, Arbiter acceleration com-

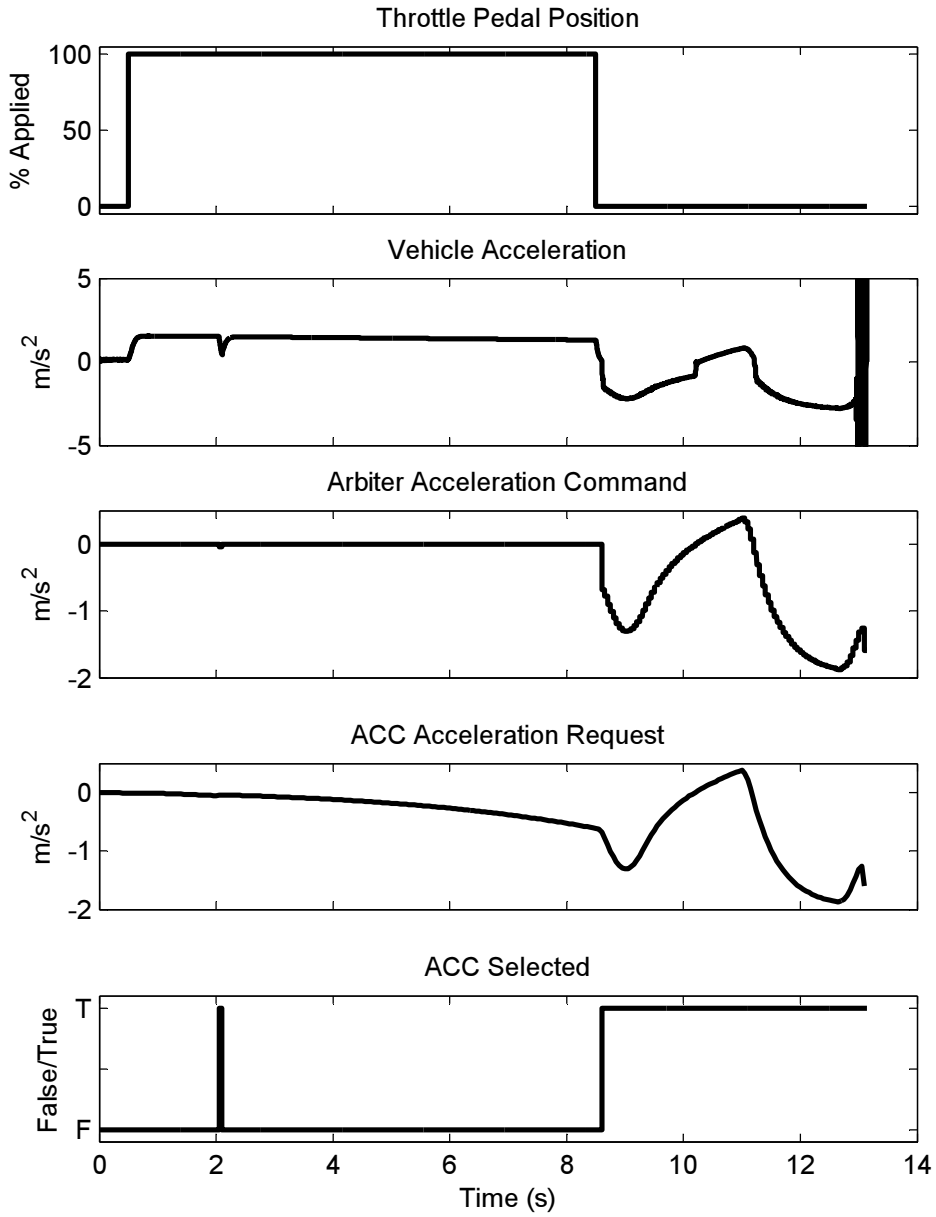


Figure 5.8. Scenario 4: ACC is engaged while the driver is applying the throttle pedal. ACC briefly takes control of vehicle acceleration, but loses control again until the driver releases the throttle pedal. ACC decelerates, then accelerates the vehicle twice before the simulation terminates.

mand, ACC acceleration request, and ACC selected over the course of the simulation. The driver begins to apply the throttle pedal at time 0.5 s. At time 2 s, the ACC is engaged, and its target speed is set to the vehicle speed. ACC gains control of the vehicle acceleration for 50 ms, the duration of one state in its internal state machine (the state duration of the system state machine is 1ms). Because the driver continues to press the throttle pedal, ACC then releases control of acceleration back to the driver on the next cycle. Although this behavior briefly violates the safety goal that allows the driver to override control in the forward direction, it is probably the desired behavior for an adaptive cruise control system. Thus, for this particular situation, the safety goal may be too restrictive.

Later, at simulation time 8.5 s, the driver releases the throttle pedal and the ACC gains control of the vehicle acceleration command. Note that ACC continues to produce acceleration requests during the time it is not selected by the Arbiter to control the acceleration command. When ACC gains control, it briefly cycles through deceleration and acceleration requests until at time 12.958 s the vehicle acceleration begins to spike erratically between very low and very high values. ACC's requested acceleration drops suddenly at time 13.1 s.

The sudden, significant changes in vehicle acceleration are likely due to wheel slip, though it is unclear why this would occur. Wheel slip, combined with a deceleration request from ACC, is likely to have caused the vehicle speed sensor to read a negative value, thus causing the goal 9 and its subgoals 9A for the Arbiter and 9B for ACC to be violated. It is clear at this point that the vehicle is in an unsafe state, regardless of the cause.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- ACC appears to briefly engage, even when the driver continues to press the accelerator pedal. This behavior could be modified to satisfy safety goal 5B by allowing

the driver to set the ACC set speed while pressing the throttle pedal without allowing ACC to become engaged at that point (i.e., record the set speed value but do not request acceleration yet). Alternatively, the goal could be modified to allow this specific behavior.

Run-time monitoring in this scenario also revealed the following additional information about the goals obtained from ICPA:

- Some subgoals violations may be caused by error in a monitored state variable in the goal, such as the vehicle acceleration sensor, rather than by a defect in an agent's control of another state variable.
- When subgoals are made more restrictive, there may be some situations that you want to exclude. These may be handled by goal realizability tactic *Split Lack of Monitorability/Controllability by Case* [46].

5.4.5 Scenario 5: throttle pedal applied, ACC engaged, CA enabled, brake pedal applied, slow vehicle in path

Description. The host vehicle is traveling forward, starting 25m behind another vehicle that is traveling between 10-25 k/h (2.78-6.94 m/s). The driver accelerates the host vehicle by applying the throttle pedal from time 0.5 s to 2.5 s. At time 2.0 s, ACC is engaged by the driver. The driver applies the brake pedal from time 4.0 s to time 9.0 s. As there are neither subsequent 'go' signals nor throttle pedal applications by the driver, the host vehicle is expected to remain stopped.

Results. Table D.5 lists the goal violations for the fifth scenario. After the vehicle is brought to a stop by the driver's braking action, it remains stopped until the simulation terminates normally at time 20 s.

Goal 1, the acceleration threshold, was violated six times between time 2.641 s and 2.731 s, with durations ranging from 5-9 ms. None of its corresponding subgoals were also violated. Goal 2, the jerk threshold, was violated 28 times between time 2.638 s and 3.41 s, with durations raging from 1-8 ms. Goal 2A was violated 28 times between time 2.651 s and 4.001 s, each with a duration 1 ms. Goal 2B was violated 30 times by ACC/LCA between time 2.05 s and 4 s, and by PA at times 0.01 s and 4.123 s, each for 1 ms. Goal 4B, which prohibits autonomous acceleration requests from a stopped position, was violated by ACC/LCA at time 5.916 s for 103 ms and at time 6.302 s for 48 ms. It was, once again, also violated by PA at time 0.01 s for 167 ms. Goal 5, which allows the driver to override all autonomous acceleration except hard braking events, was violated at time 2.052 s for 50 ms and at time 4.02 s for 50 ms. Its subgoal 5A was violated at time 2.051 s for 50 ms and at time 4.001 s for 50 ms. Subgoal 5B was violated once at time 2 s for 50 ms.

A closer look at acceleration and jerk for the vehicle, the Arbiter's acceleration command, and ACC's acceleration request, as well as the ACC_Selected tag are shown in Figure 5.9. ACC is granted control of the acceleration command at time 2.601, after the throttle pedal is released at time 2.5 s. At this time, some residual jerk from the release of the throttle pedal is still being experienced by the vehicle. Thus, the vehicle-level safety goal is violated, even though the jerk was not necessarily caused by ACC. The gradual changes in acceleration command between times 2.601 s and 4.01 s cause many 1 ms violations of its jerk threshold. Once again, this is an artifact of sampling, and are likely not signs of a significant problem.

As in Scenario 4, ACC gains control of acceleration for one state when it is enabled at time 2.0 s. ACC also stops requesting control when the brake pedal is applied at time 4.0 s. Although ACC updates on a 50 ms cycle, the Arbiter updates its acceleration commands and acceleration source tags every 1 ms. The Arbiter is not immediately releasing control

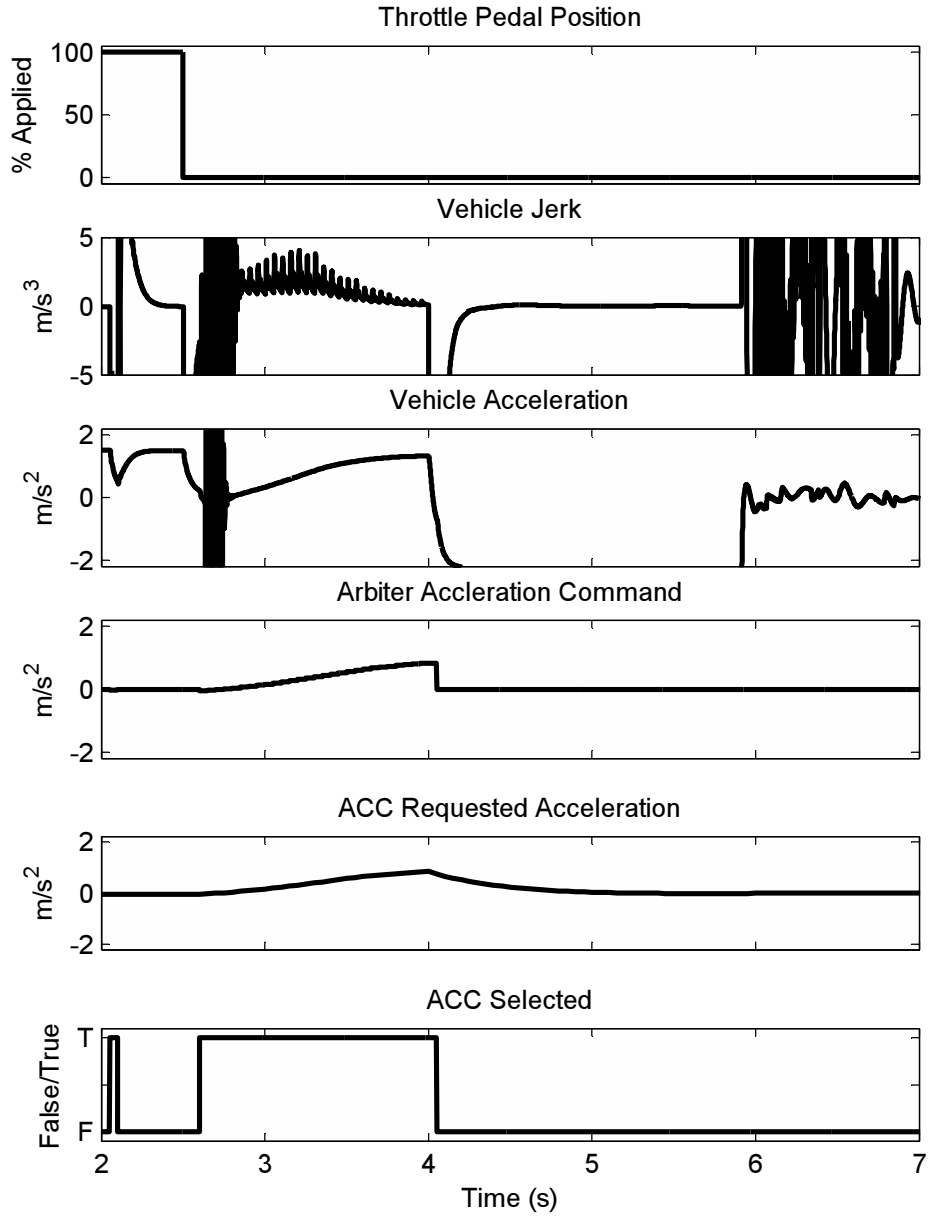


Figure 5.9. Scenario 5: The driver releases the throttle pedal. Control of acceleration is gained by ACC 0.101 seconds later.

of acceleration when a brake pedal is applied, rather it is waiting for ACC to stop requesting acceleration. The difference in state transition times between the two causes the Arbiter's goal to be violated for the duration of one ACC state.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- The Arbiter appears to not be enforcing subgoal 5A. Rather, it relying on the feature subsystems to enforce subgoal 5B.
- Different subsystems have different state transition times. This may complicate goal satisfaction if a subsystem updates controlled state variables at a slower rate than the overall system state period.

Run-time monitoring in this scenario also revealed the following additional information about the goals obtained from ICPA:

- Transitions of control from one agent to another may have unavoidable residual delays in jerk and acceleration that cause goals restricting those values to be violated. More work needs to be done understanding what durations of jerk and acceleration thresholds are acceptable in these situations.

5.4.6 Scenario 6: throttle pedal applied, ACC engaged, CA enabled, LCA engaged, slow vehicle in path

Description. The host vehicle is traveling forward, starting 25m behind another vehicle that is traveling between 10-25 k/h (2.78-6.94 m/s). The driver accelerates the host vehicle by applying the throttle pedal from time 0.5 s to 2.5 s. At time 2.0 s, ACC is engaged by

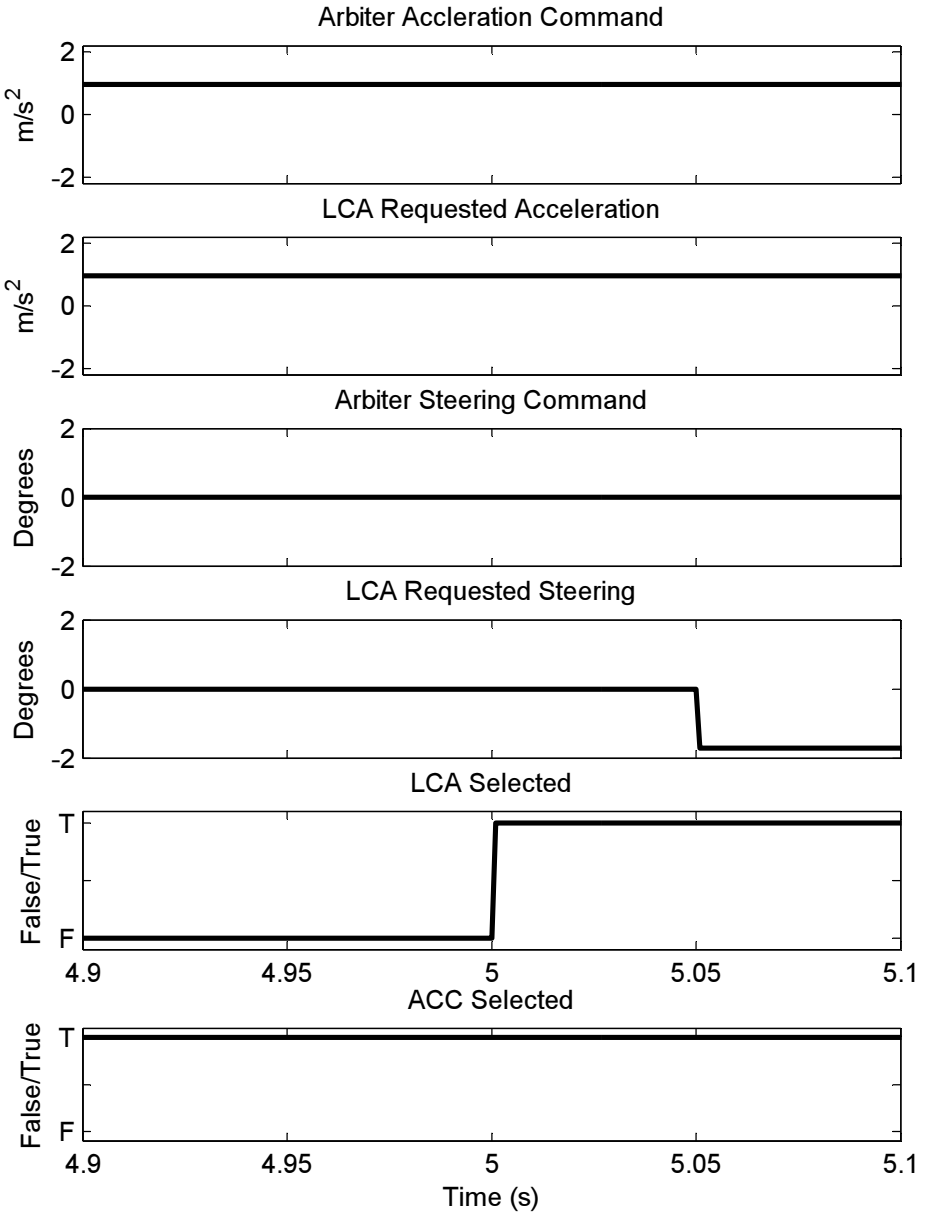


Figure 5.10. Scenario 6: LCA is enabled at time 5.0 s, and gains control of acceleration and steering at time 5.001 s. At time 5.051, LCA requests steering, but the steering command remains unchanged.

the driver. LCA is enabled by the driver at time 4.0 s, and engaged at time 6.0 s. LCA is expected to perform the lane change maneuver as requested.

Results. Tables D.6 and D.7 list the goal violations for the sixth scenario. After LCA is engaged by the driver, the vehicle continues moving in the same path. ACC attempts to slow the vehicle behind the slower lead vehicle, but the simulation terminates early at time 13.954s.

Goal 1, the acceleration threshold, was violated 17 times, between time 2.641 s and 13.953 s, with durations ranging from 1 ms to 9 ms. As with all prior simulations, none of its subgoals were violated. Goal 2, the jerk threshold was violated 108 times, between time 2.651 s and 13.951 s. Subgoal 2A was violated 83 times from time 2.651 s and 13.951 s. Subgoal 2B for ACC/LCA was violated 85 times, from time 2.05 s to time 13.95 s. Goal 3 and the identical subgoal 3A, acceleration and steering agreement, was violated at time 5.052 for 8.902 sec. Goals 5, 5A, and 5B were again violated when ACC was engaged at time 2 s. Also as before, PA violates subgoal 2B at times 0.001 s and 4.12 s, and subgoal 4B at 0.01 s for 167 ms. Goals 9, 9A, and 9B (for ACC and LCA), which prohibits subsystems from controlling acceleration and steering outside their intended vehicle directions, were all violated at time 13.905 s.

A plot of the Arbiter's acceleration command and steering command, LCA's requested acceleration and requested steering, and the LCA_Selected and ACC_Selected tag are shown in Figure 5.10. LCA is enabled at time 5.0 s, and is granted control at time 5.001 s. In the system design, LCA is only enabled when ACC is also enabled and engaged. When the LCA_Selected tag is set, the ACC_Selected tag remains set. This means that two separate features, one that requests no steering and the other that requests no acceleration, combine to form the lane change functionality. However, when LCA changes its steering request at time 5.051 s, the Arbiter's steering command does not change, violating the accelera-

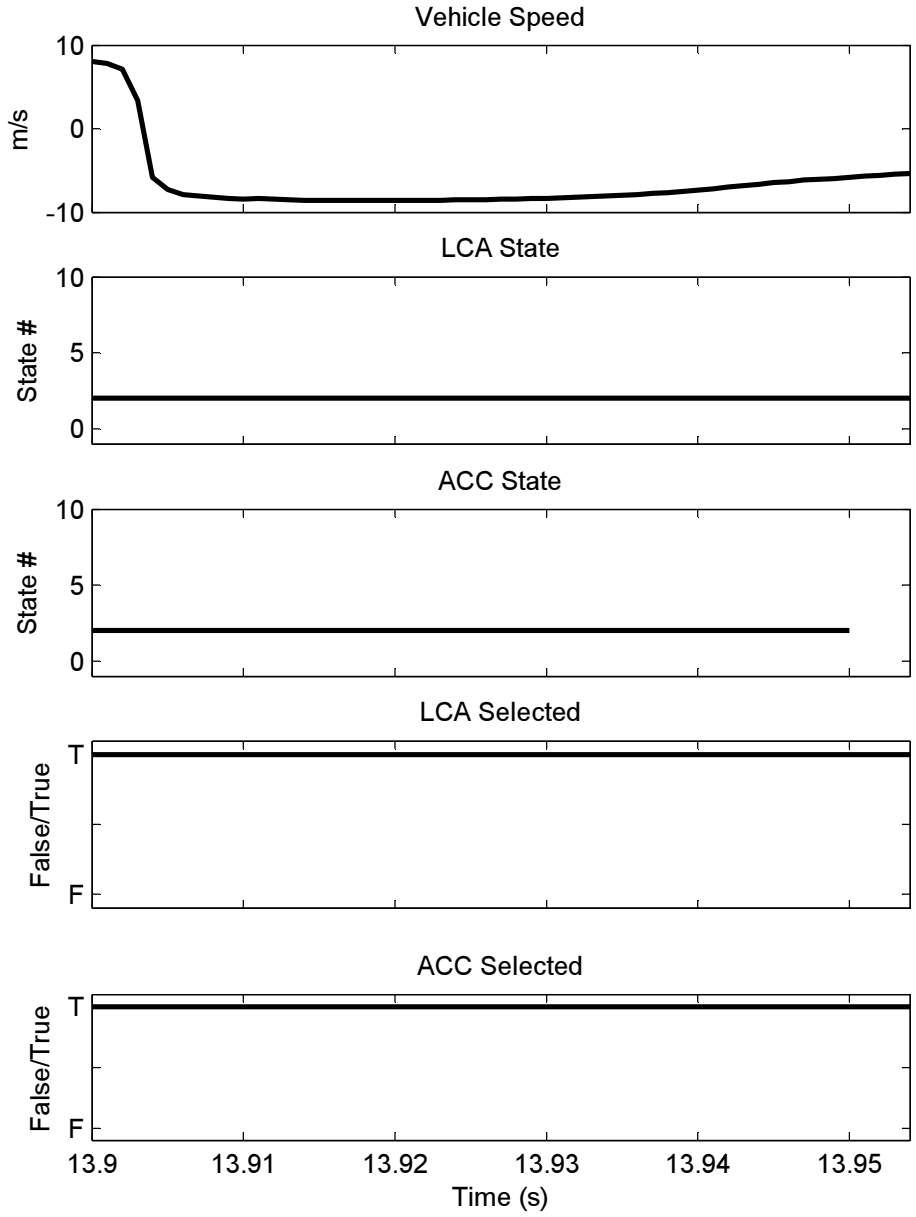


Figure 5.11. Scenario 6: Vehicle speed becomes negative, LCA and ACC are still active and selected to control vehicle acceleration.

tion/steering agreement subgoal.

A plot of the vehicle acceleration, acceleration command, and ACC/LCA acceleration request is shown in Figure 5.11. Even though the vehicle speed has become negative, indicating the vehicle is moving in the reverse direction, ACC and LCA continue to request control, and the Arbiter continues to provide it. This results in violation of goal 9 and subgoals 9A and 9B. It is unclear why the vehicle speed suddenly becomes negative. Once again, this could be due to wheel slip during deceleration. If this is the case, it may be desirable to allow the feature subsystem causing the deceleration to continue. However, the simulation does terminate shortly after the vehicle speed becomes negative. This may be due to the way the vehicle subsystems are handling this system condition.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- LCA is not given control of steering, even though the Arbiter selected LCA for control. At the time of the study, LCA was less developed than ACC or CA, so this may be simply a result of incomplete implementation or subsystem integration.
- The different tags indicating which subsystem is selected may a source of confusion with regards to the acceleration/steering agreement goal. If both ACC and LCA are selected, perhaps the ACC's steering request (i.e., no steering) is overriding LCA's steering request.

Run-time monitoring in this scenario also revealed the following additional information about the goals obtained from ICPA:

- Goals that prohibit certain features from gaining control of the vehicle when vehicle is moving in the forward or reverse direction may need to be examined further to consider what is the proper behavior when wheel slip occurs.

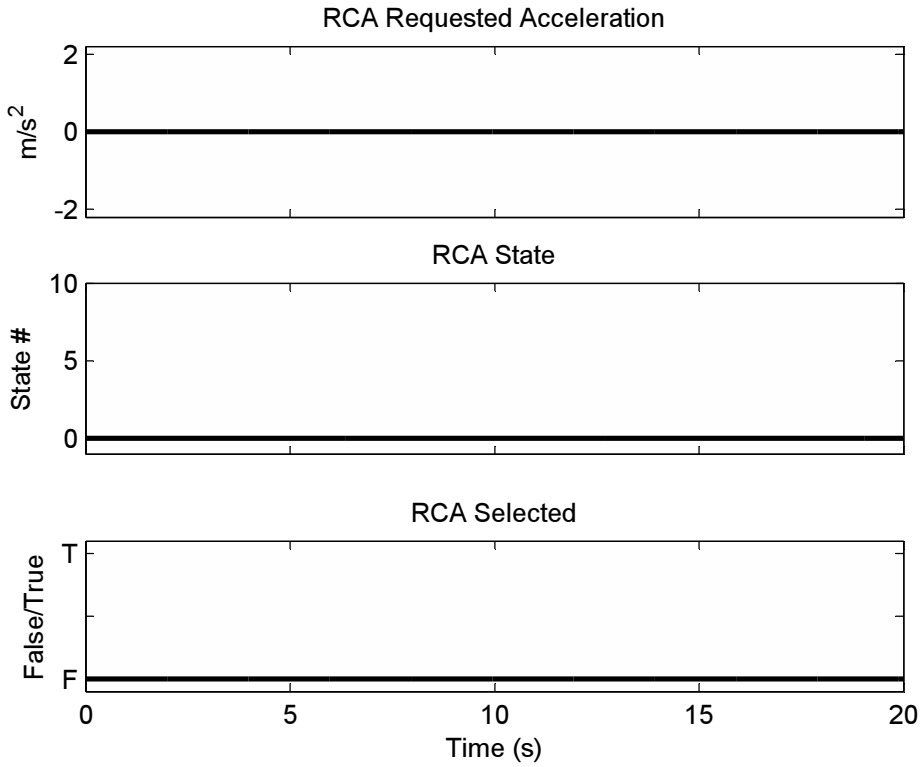


Figure 5.12. Scenario 7: RCA is enabled at the simulation start, but never engages to stop the host vehicle before reaching the stopped vehicle behind it.

5.4.7 Scenario 7: in reverse, RCA enabled, stopped vehicle in path

Description. The host vehicle is traveling in reverse, 30 m in front of another stopped vehicle, with RCA enabled. RCA is expected to perform a hard braking action to stop the host vehicle.

Results. Table D.8 lists the goal violations for the seventh scenario. The vehicle makes no attempt to stop and 'collides' with the parked vehicle in its rear path.

There were no goal violations for the vehicle. PA behaved as before in the previous scenarios, and continued to output acceleration requests, violating subgoal 2B at time 0.001 s and 6.264 s, each for 1 ms. Because PA's acceleration requests began when the vehicle

was still stopped, its subgoal 4B was also violated, for a duration of 44 ms. ACC and LCA both also sent out acceleration requests, even though neither was enabled, causing 42 jerk violations from 8.45 s to 15.6 s, each 1 ms in duration.

Figure 5.12 shows RCA's acceleration request, internal state, and RCA_Select tag. RCA never engages to stop the host vehicle. This is most certainly due to an unfinished implementation.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- ACC, LCA, and PA send out acceleration requests continuously, not only when they are disabled, but also when the vehicle is traveling in the reverse direction.
- RCA implementation is not complete enough to evaluate.

Run-time monitoring in this scenario also revealed the following additional information about the goals obtained from ICPA:

- There were no vehicle-level goal violations, even though the vehicle 'collided' with the vehicle in its path. If the simulation environment were able to simulate vehicle collisions, some vehicle-level violations might occur. However, any monitored values that would be detected from such a situation may only provide information for post-accident analysis, not prevention.

5.4.8 Scenario 8: in reverse, ACC engaged, stopped vehicle in path

Description. The host vehicle is traveling in reverse, 30 m in front of another stopped vehicle. ACC is enabled and engaged by the driver at time 2.0 s. The host vehicle is expected to ignore the driver's request and continue moving backward.

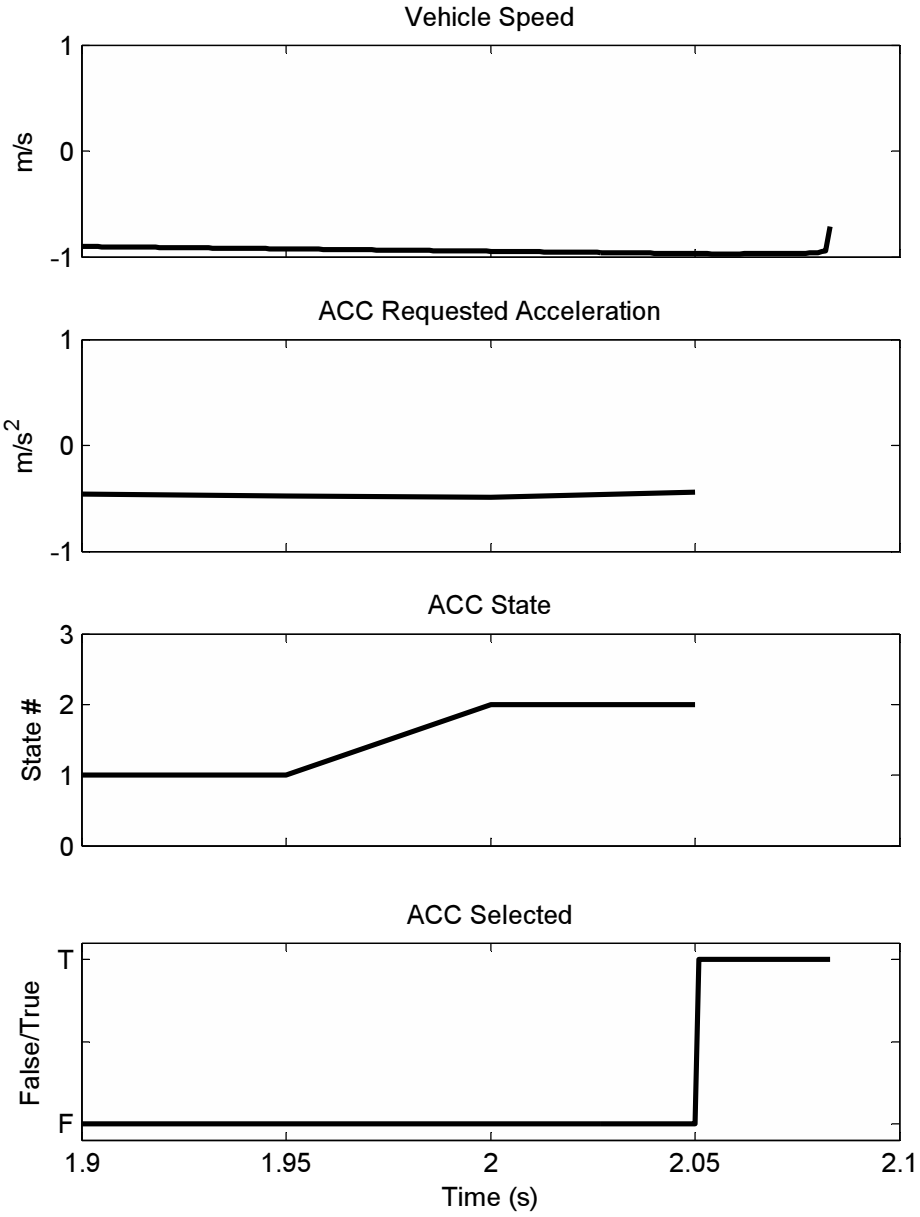


Figure 5.13. Scenario 8: After ACC is engaged at time 2.0 s, it is selected as the source of the acceleration command at time 2.05 s.

Results. Table D.9 lists the goal violations for the eighth scenario. Shortly after ACC is engaged, the simulation terminates early.

In this scenario, Goal 1, the acceleration threshold, is violated once at time 2.08 s, until termination of the simulation 3 ms later. None of its subgoals were violated. Goal 2, the jerk threshold, is violated at time 2.055 s for 8 ms, at time 2.07 s for 11 ms, and at time 2.083 s for 1 ms. Subgoal 2A was not violated, but subgoal 2B was for ACC/LCA at time 2.05, just after they were engaged. Once again, subgoals 2B and 4B were violated for PA at the start of simulation. Goal 9 and subgoal 9A, restriction on subsystem control in the backward direction, were violated at time 2.051 s for 32 ms. Subgoal 9B was violated by ACC at time 2 s for 83 ms.

Figure 5.13 shows the vehicle speed, ACC's acceleration command, ACC's state, and the ACC_Selected flag. After ACC is engaged at time 2.0 s, the Arbiter selects ACC to be in control of acceleration at time 2.05 s. The simulation terminates in error 33 ms later.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- ACC can be engaged by the driver when the vehicle is moving in reverse. This should not be allowed
- The Arbiter selects ACC as the source of acceleration commands when the vehicle is moving in reverse. This also should not be allowed.

Run-time monitoring in this scenario also revealed the following additional information about the goals obtained from ICPA:

- Goal redundancy does not protect against failure of both subsystems to satisfy the goal.

5.4.9 Scenario 9: stopped, PA engaged, stopped vehicle in path

Description. The host vehicle is stopped 20 m behind another stopped vehicle. PA is enabled and engaged by the driver at time 2.0 s. PA is expected to initiate a parking action.

Results. Table D.10 lists the goal violations for the ninth scenario. The simulation terminates early 113 ms after PA is engaged by the driver.

Goal 1, the acceleration threshold, was violated at time 2.098 s for 15 ms. Goal 2 was violated six times between time 2.003 s and 2.112 s, for durations ranging from 1 ms to 46 ms. Its subgoal 2B was violated by park assist again at time 0.001 s for 1 ms. Goal 3 and 3A, acceleration and steering agreement, was violated at time 2.001 s for 112 ms. Subgoal 4B was again violated by park assist at time 0.01 s.

Figure 5.14 shows the vehicle speed, vehicle acceleration, Arbiter acceleration command, PA acceleration request, PA state, and PA_Selected flag. At time 2.001 s PA is selected as the source of the acceleration command, but the acceleration command is not set to the PA acceleration request. This causes Goal 3 and subgoal 3A to be violated, and is likely to be a contributing factor to the early termination at time 2.113 s.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- When PA is engaged while the vehicle is stopped, it is selected as the source of vehicle acceleration and steering, but its acceleration request is not used to determine the Arbiter's acceleration command.

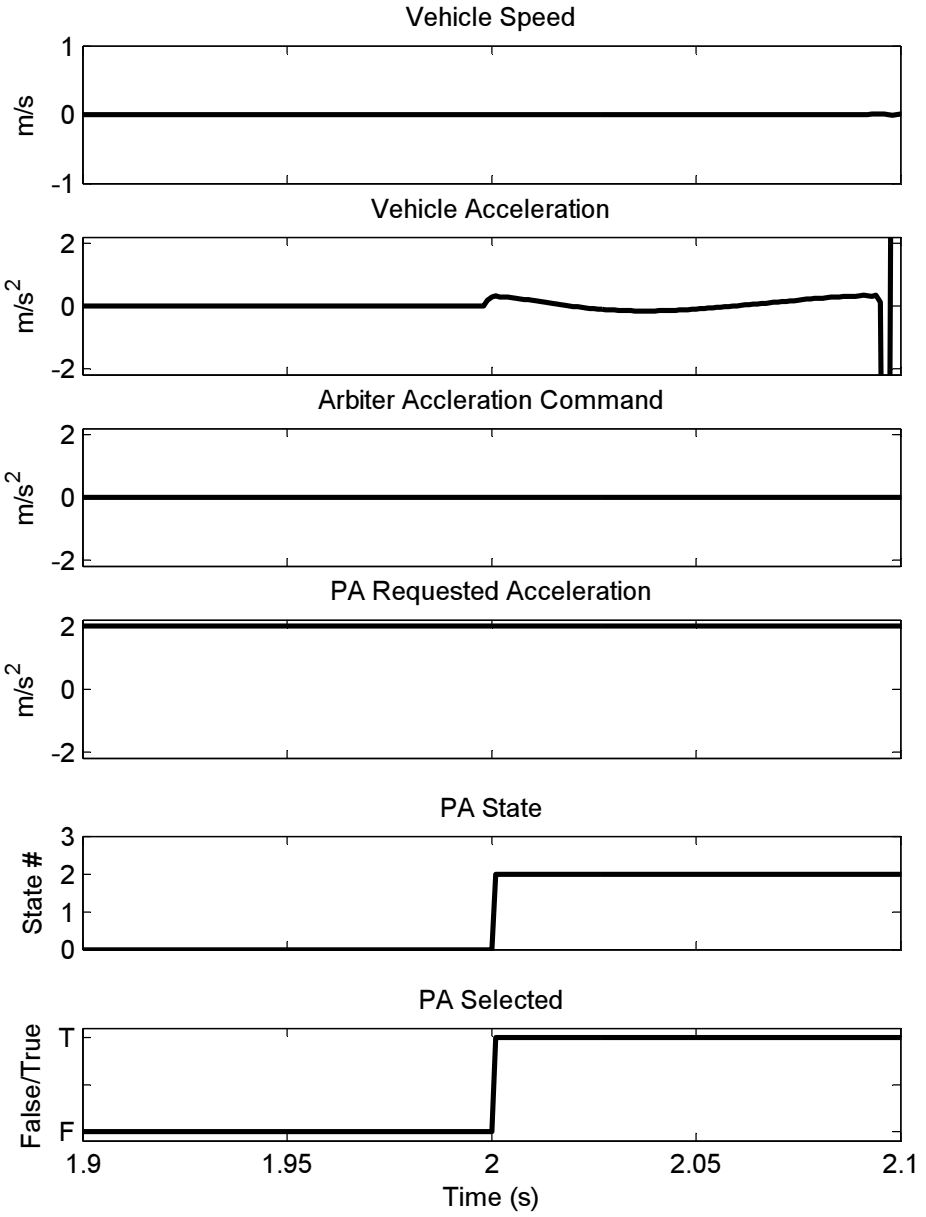


Figure 5.14. Scenario 9: When PA is engaged, it is selected as the source of the acceleration command, but the acceleration command is not equal to the PA acceleration request.

5.4.10 Scenario 10: stopped, ACC engaged, stopped vehicle in path

Description. The host vehicle is stopped 20 m behind another stopped vehicle, ACC is enabled and engaged by the driver at time 2.0 s. ACC may either ignore the request or accelerate the vehicle. If the driver initiates ACC control, then acceleration is acceptable for safety goal 4. However, it is also safe for the vehicle to remain stopped.

Results. Table D.11 lists the goal violations for the tenth scenario. Although the driver attempts to engage ACC, it neither enters an engaged state nor controls the acceleration command. However, the vehicle begins to accelerate anyway, until CA engages to try to stop the host vehicle before hitting the stopped vehicle in its path. The simulation terminates early at 14.625 s during this hard brake attempt from CA.

Goal 1 is violated once at time 14.589 for 4 ms. None of its subgoals were violated. Goal 2, the jerk threshold, is violated at time 14.583 s for 8 ms and at time 14.598 s for 2 ms. Its subgoal 2B was violated once for CA at time 14.6 s, for a duration of 1 ms. Once again, PA experienced violations of subgoals 2B and 4B at time 0.01 s.

Figure 5.15 shows vehicle speed, vehicle acceleration, the Arbiter's acceleration command, the ACC acceleration request, the ACC state, and the ACC_Selected flag. Although the driver attempts to engage ACC while the vehicle is stopped, ACC does not enter an active state, and the Arbiter does not select ACC for the acceleration command. This is within the bounds of safe system behavior. However, the vehicle begins to accelerate, and it is unclear where the source of that acceleration is. Perhaps the ACC set speed is being used, even though ACC is not requesting it to be.

Summary. Run-time monitoring in this scenario revealed the following additional information about the semi-autonomous automotive system:

- Even though ACC is neither active nor selected as the source of the acceleration

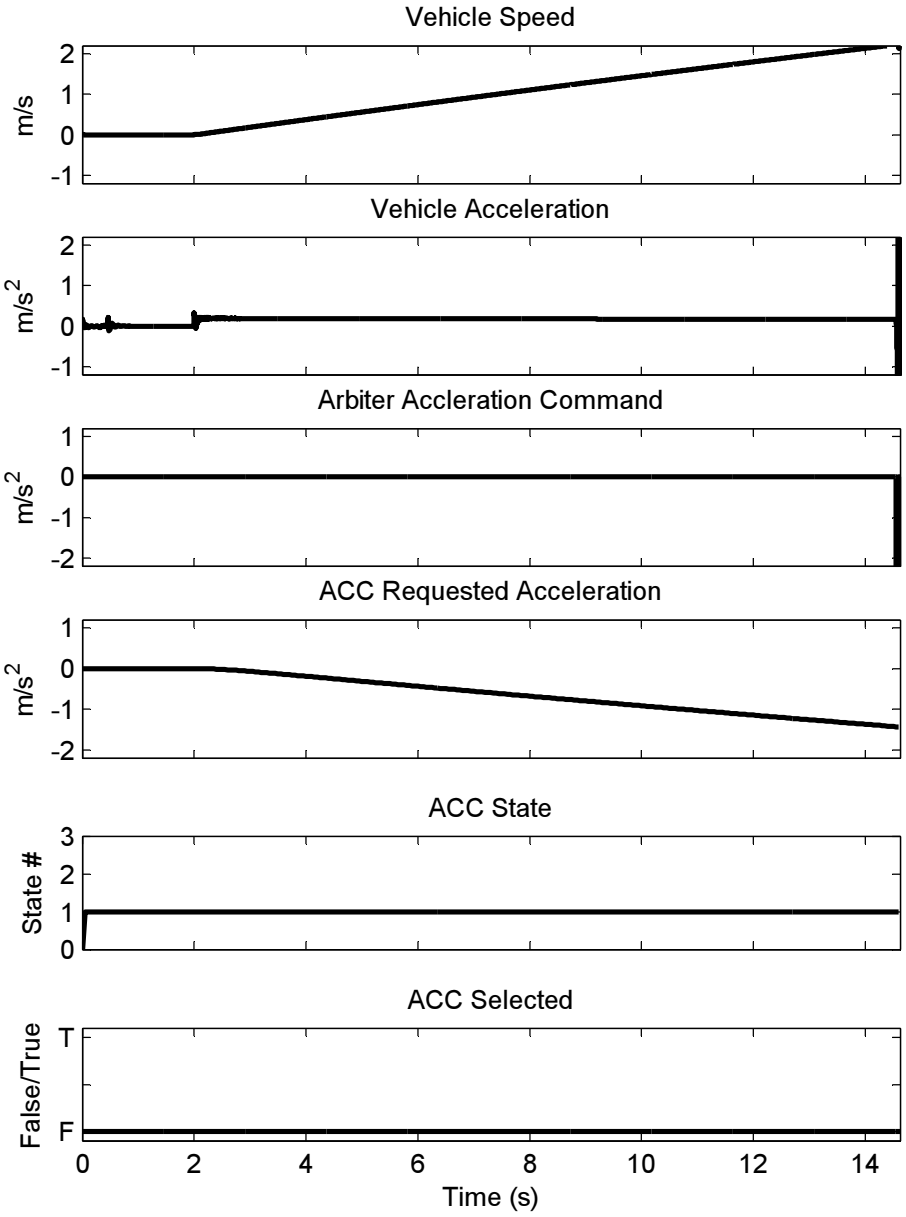


Figure 5.15. Scenario 10: When the driver attempts to engage ACC at time 2.0 s, ACC does not become active, nor is it selected by the Arbiter to control steering. The vehicle, however, does begin to accelerate.

command, the vehicle begins to accelerate when the driver attempts to engage ACC.

Run-time monitoring in this scenario also revealed the following additional information about the goals obtained from ICPA:

- Perhaps there is some relationship between the driver's request to engage ACC and vehicle acceleration that bypasses the Arbiter and ACC. This indicates the indirect control relationships defined in ACC are incorrect or incomplete.

5.5 Conclusion

ICPA was applied to nine system safety goals for a semi-autonomous automotive system developed in a commercial automotive research lab. The goals and resulting subgoals were then monitored at run-time in ten different driving scenarios. Although not all goal violations were detected in the subgoal monitors, subgoal monitoring did reveal a number of design defects in the incomplete system implementation, sometimes when the hazard was not yet visible at the system level. Monitoring at both the system and subsystem level revealed some goal violations that may not be detectable by the driver. These results are discussed in more detail in Chapter 6.

Chapter 6

Discussion

Even though the automotive system under review was a research system and not a commercial product, it was a real automotive vehicle. Subsequent implementations of the system included hardware-in-the loop simulation and a drivable test vehicle. There are several advantages to using a real system built in a commercial lab instead of a toy system developed by an academic lab. First, the system under review in this thesis was designed and built by application domain experts. This makes the system more representative of commercial vehicles.

Second, although information about the system learned from ICPA was passed back to development, ICPA and monitoring were done separately from design and development. That is, the same engineers who built the system did not perform the ICPA and monitoring. In addition, the system was incomplete when ICPA was performed. The system was relatively functional, but certainly incomplete. These factors combined to provide a platform that had real design defects, not ones that were added just for ICPA, but was functionally complete enough to do run-time testing.

6.1 Lessons learned about the system

Some design issues were discovered during review of the system requirements and design for ICPA, such as the reverse arbitration logic. Others were discovered during monitoring.

This section describes the information learned about the system under review from the ICPA and goal and subgoal monitoring.

6.1.1 Desired behaviors confirmed

ICPA and subgoal monitoring confirmed the following desired behaviors in the system under review:

1. Goal redundancy in the Arbiter appears to block the improper acceleration requests from PA, as intended. This was confirmed in all scenarios, except Scenario 9, when PA was engaged and expected to take control of the vehicle.

2. CA appears to continue to execute a hard brake when the driver applies the throttle pedal. This is the intended behavior of the system, and was confirmed in Scenario 3.

3. The RCA and LCA implementations are not complete enough to evaluate. This was discovered in Scenario 6 and Scenario 7

6.1.2 Problems identified

ICPA and subgoal monitoring identified the following specific problems in the system under review:

1. Prioritization of feature subsystem control requests in steering arbitration is the reverse of the prioritization in the acceleration arbitration. This defect was discovered during the ICPA process and confirmed by subgoal monitoring when it caused 3A to be violated in Scenario 2. This particular subgoal was assigned as a single responsibility subgoal to the Arbiter, so no subgoal violations occurred for the feature subsystems. In addition, it was discovered that its parent goal 3 cannot actually be

monitored in the subsystem. Thus, subgoal monitoring is the only way to identify when violations of the goal occur.

2. PA is sending out acceleration requests when it is neither engaged nor even enabled. This caused subgoal 4B to be violated in every scenario. As mentioned above, the Arbiter appears to be blocking these commands, so the redundant subgoal 4A and the parent goal 4 are not violated.

3. ACC and LCA often send out acceleration requests regularly, not only when they are disabled or enabled but not engaged, but also when the vehicle is traveling in the reverse direction. This causes goal 4B to be violated in scenario 5 for both.

4. The vehicle violates goal 1, the acceleration limit for autonomous control, shortly before the simulation terminates in error in 7 out of 10 scenarios. This sudden spike in acceleration may be caused by wheel slip. Early termination may occur because the vehicle has entered an unsafe state, or may simply signify that the system implementation does not handle these type of extreme sensor values. When this occurs, sometimes the vehicle direction suddenly changes. This causes subgoal 9A for the Arbiter and 9B for ACC to be violated in scenarios 4, 6, and 8. Although in this instance the subgoals are violated due to some error in the speed sensor or wheel slip, and not necessarily due to the subsystem whose goal was violated, the subgoals do identify an unsafe vehicle state.

5. CA appears to execute the hard brake in a manner that is intermittent, rather than continuous. All CA braking scenarios resulted in early termination of the simulation, or a ‘collision’ with the vehicle in the forward path. Each time CA releases and then re-engages the brake, Goal 2, the autonomous jerk threshold, is violated for longer

than one state in scenario 3. Subgoal 2A for the Arbiter and 2B for CA are also violated, but only for one state.

6. If ACC is engaged by the driver while the driver is also accelerating, ACC engages for one state, before releasing control back to the driver. This violates subgoals 5A for the Arbiter and 5B for ACC in scenarios 4, 5, 6, and 8. This behavior could be modified to satisfy the safety goal by allowing the driver to set the ACC set speed while pressing the throttle pedal without actually allowing ACC to become engaged at that point (i.e., record the set speed value but do not request acceleration yet). Alternatively, the goal could be modified to allow this specific behavior.

7. The Arbiter appears to not be enforcing subgoal 5A. Rather, it is relying on the feature subsystems to satisfy 5B. This is seen when subgoals 5A and 5B are violated in scenarios 4, 5, 6, and 8.

8. LCA is not given control of steering when the Arbiter has selected LCA for control. This violates goal 3 in scenario 6. At the time of the study, the implementation of LCA was less developed than ACC or CA, so this may be simply a result of incomplete implementation or subsystem integration.

9. ACC can be engaged by the driver when the vehicle is moving in reverse. This causes goal 9B to be violated in scenario 8.

10. The Arbiter appears to not be enforcing subgoal 9A. It selects ACC as the source of acceleration commands when the vehicle is moving in reverse and ACC is engaged. This causes subgoal 9A to be violated in scenario 8.

11. When PA is engaged while the vehicle is stopped, it is selected as the source of vehicle acceleration and steering, but its acceleration request is not used to determine the

Arbiter's acceleration command. This causes subgoal 3A to be violated in scenario 9.

These defects might be identified in regular system test. In testing, if the simulation terminates in error or a simulated collision occurs, the design and simulation data would be analyzed to identify the root cause of the fault. In these situations, safety goal and subgoal violations aid root cause analysis by immediately identifying certain state variables that may be related to the incorrect behavior.

Sometimes the feature subsystem subgoals are violated but faulty system behavior would not be readily apparent to system testers, either because the fault is so transient it goes unnoticed, such as when the ACC engages while the driver is still pressing the throttle pedal, or because the subgoal violation is masked by goal redundancy in the Arbiter, such as when PA, ACC and LCA continue to send acceleration requests without being engaged or active.

6.1.3 General design insights

Applying ICPA and monitoring of goals and subgoals also led to the following insights about design and implementation of this type of system, in general:

1. Divided arbitration of longitudinal acceleration and steering can complicate goals that restrict coordinated behavior between the two types of vehicle control.

2. Using separate flags to indicate when a subsystem is selected for control makes it possible to attribute control actions to multiple sources. This may make the actual source of acceleration and steering commands ambiguous to subsystems that use that information.

3. A centralized Arbiter is a potential source of single-point failures of the system.

4. Using differing state transition times in different subsystems may complicate goal satisfaction if a subsystem updates controlled state variables at a slower rate than the overall system state period.

5. It is probably safer to not output any control requests when the subsystem is not engaged, even if flags are used to indicate whether or not the request is active.

By discovering this information during system development, these issues can be corrected prior to integration testing. After the system is complete, run-time monitors may be able to provide diagnostic services for safety-related system and subsystem errors.

6.2 Lessons learned about ICPA

In addition to finding specific design defects in the system design and implementation, the process of applying the ICPA and monitoring the goals and subgoals also provided insight into the ICPA itself. The following information was learned about the ICPA and subgoals it produced:

1. Monitoring of the goals and subgoals during subsystem and integration testing can help identify potential defects in the design and implementation. The defects found in this analysis are listed in Section 6.1.2.

2. Some goals can only be monitored at the subsystem level. If the goal restricts control of a state variable directly controlled by some agent in the system, such as a command to an actuator or a network message, then the highest level in the system hierarchy at which the goal can be monitored is the level containing the subsystem that controls the variable. If the goal restricts two or more state variables directly controlled by different agents in the system, or one or more state variable that is sensed from system

dynamics or the environment, then the goal is monitored independently from a single subsystem. This is applicable to any system with closed-loop control of physical actuation and sensing. For example, vehicle goal monitors for goals 3, 5, 6, 7, 8, and 9 are the same as the Arbiter subgoal monitors, because they restrict which subsystem is controlling acceleration or steering. This cannot be detected by a sensor in the same manner as the actual vehicle acceleration or jerk, but must be read from the flag set by the arbiter.

3. When a subgoal is violated at one level, violations in subsystem goals may be able to point toward a root cause. For example, when ACC is engaged by the driver as the vehicle is traveling backward, the subgoal violation in 9B is also seen in the violation of subgoal 9A.

4. In any system that employs a redundant goal coverage strategy, monitoring all the subgoals as well as the safety goal can help identify when inappropriate subsystem behaviors are being masked by other subsystems. Subgoal 4B is violated by ACC and PA whenever the vehicle is stopped because those subsystems send acceleration requests when they are not engaged. However, subgoal 4A and goal 4 are not violated.

5. Almost all safety subgoals are restrictive, often in multiple ways. Most goal restriction comes from variability, or jitter, in the values monitored or controlled by the system agents. Any system with physical actuation or sensing will experience this variability, and will require restrictive subgoals. All goals analyzed with ICPA in this study had restrictive subgoals. For example, all feature subsystems restricted their acceleration and steering requests to meet the threshold, whether or not they were selected as the winner of arbitration. If a feature subsystem is not selected as the

winner of arbitration, its requests that violate the acceleration threshold will not have an affect on the vehicle acceleration. However, it is easier to implement the more restrictive behavior.

6. Goal redundancy along a single indirect control path in a system's behavioral hierarchy can only protect against defects that occur in subsystems located earlier in the control flow (i.e., more removed from the actual source of direct control). Goal redundancy does not protect against defects in the subsystems located later in the control flow, where single-point failures may still occur. For the particular automotive system used in this evaluation, this means that the arbiter can guard against failures in the feature subsystems, but is itself a source of single-point failures (as seen in scenario 6 when it selects LCA as the winner of arbitration, but does not actually use the steering request from LCA to control the vehicle). An example of a type of goal redundancy at the same level in the control hierarchy is the elevator control system from Chapter 4. In that example, the drive controller and emergency brake have indirect control paths from the elevator position state variable.

7. The jerk threshold goal is too restrictive to be implemented practically. Every time a different acceleration is requested by a feature subsystem or commanded by the Arbiter, the jerk threshold subgoals are violated for at least one state. This occurs in every scenario except scenario 7, where no subsystem engaged to control the vehicle because RCA was incomplete. A revised goal would allow jerk thresholds to be violated for a single state. This might be true of any type of digital control system.

8. Transitions of control from one agent to another may have unavoidable residual delays in jerk and acceleration that cause goals restricting those values to be violated. More work needs to be done understanding what durations of jerk and acceleration

thresholds are acceptable in these situations. This residual jerk and acceleration is likely to occur in any system with physical actuation and sensing. This occurs in scenario 3, when CA engages and applies and releases the brake four times.

9. Goals 1 and 2 should be split by case, where the goal for forward motion is different from the goal for backward motion (as was done for goals 5-6 and 8-9). This did not actually cause any goal or subgoal violations in the evaluation because RCA never engaged as it should to stop the vehicle in reverse. This was just something that came up while uncovering the root cause of some violations of subgoals 5 and 9.

10. Goal 1 is not fully composed by subgoals 1A and 1B. The acceleration threshold for autonomous control was exceeded just prior to early termination of the simulation in scenarios 1, 2, 4, 6, 8, and 9. This may indicate the indirect control relationships used to define acceleration response that were used in the ICPA are incorrect, or that the vehicle has entered an unsafe state due to wheel slip.

11. Some subgoals violations may also be caused by an error or exceptional value in a monitored state variable in the goal, such as the vehicle acceleration sensor that experience wheel slip, rather than by a defect in an agent's control of another state variable. In scenarios 4, 6, and 8, the sudden spike in acceleration and violations of subgoal 9A for the Arbiter and 9B for ACC appear to be caused by wheel slip. In the ICPA, the possibility of wheel slip was not considered when defining goals and subgoals that restrict acceleration, speed, or movement in a particular direction. Thus, these goals and subgoals were violated any time wheel slip occurred in simulation. However, wheel slip itself may indicate an unsafe system state. Issues related to intense spikes in speed and acceleration need to be examined further during ICPA for this system.

12. The nature of when CA performs its braking actions was misunderstood during the ICPA. There are some states in which CA is engaged, but not performing a braking action. These states should be excluded from the analysis when CA is active. This was seen in scenario 3, when CA continued to be selected as in control, but transitioned four times from a hard brake to a preparation state. CA failed to stop the host vehicle before reaching the stopped vehicle in its path, so this may actually be incorrect system behavior. The nature of this brake preparation state should be examined further to see if CA should continue to override the driver when CA is engaged but not performing a hard brake.

13. The indirect control relationships between the CA acceleration request, Arbiter acceleration command, and resulting vehicle acceleration do not perform as defined in the ICPA. There is a greater delay between the acceleration command and the resulting vehicle acceleration response. System goal 5 should be adjusted to account for this delay. Subgoals 5A and 5B, however, seem to be correct. In this case, monitoring revealed a parent goal that is too restrictive to be satisfied while still satisfying the functional behavior of the CA feature subsystem. This was seen in scenario 3, when CA requested a hard brake, and the Arbiter commanded a hard brake, but there was a delay in the vehicle response performing the hard brake.

14. When subgoals are made more restrictive, there may be certain operational behaviors that violate the goal but are desirable in the system. These may be handled by goal realizability tactic *Split Lack of Monitorability/Controllability by Case* [46] (i.e., have separate safety goals for each type of situation). For example, it may be desirable to allow ACC to briefly engage when selected by the driver while the driver is also accelerating. This situation caused subgoals 5A and 5B to be violated in scenarios 4,

5, 6, and 8. The goal could be modified to exclude the case when the driver presses the ACC engage button.

15. There were no vehicle-level goal violations in scenario 7, even though the vehicle 'collided' with the vehicle in its path. If the simulation environment were able to simulate vehicle collisions, some goals and subgoals may have been violated. However, any monitored values that would be detected from such a situation may only provide information for post-accident analysis, not prevention. In this scenario, RCA is not implemented completely enough to provide the needed functionality to stop the vehicle. The unsafe situation occurs, not because the vehicle is violating any of the defined safety goals, but because the feature itself is not performing its intended functionality. Even though the driver is supposed to remain active and alert in semi-autonomous automotive systems, perhaps the functional behavior of features considered to provide 'active safety,' as opposed to those that provide 'driver convenience,' should also be included in the set of system safety goals.

16. In scenario 10, there appears to be a relationship between the driver's request to engage ACC and vehicle acceleration that bypasses the Arbiter and ACC. This indicates the indirect control relationships defined in ACC are incorrect or incomplete.

These results indicate that ICPA must be an iterative process, similar to hazard analysis and requirements engineering. Subgoals defined before system design and implementation may be found to be incorrect or impractical during those stages of the system development cycle. This is true for any distributed embedded system with reasonable complexity.

6.3 Conclusion

The primary purpose of this evaluation was to determine whether or not the ICPA technique and resulting subgoals were useful. In a large-scale real automotive system, applying ICPA and monitoring the subgoals it produced uncovered eleven critical design defects while the system was still under development, which are listed in Section 6.1.2. In addition, goal and subgoal monitoring revealed some of the limitations of ICPA, including impracticality of some goal reductions, the unidirectional nature of some goal redundancy strategies, and the inability to monitor some goals at the system level. These results show that ICPA can be applied to large-scale embedded systems with closed-loop control that use physical actuation and sensing.

Chapter 7

Conclusion

System safety, as an emergent system property, is difficult to handle in an hierarchical, distributed design process because traditional decomposition tactics are not always applicable. As safety-critical embedded systems become more pervasive and more complex, a good requirements process becomes more and more essential for handling this emergence. As part of that process, design and analysis of the system safety requirements should be systematic and fully documented.

7.1 Thesis contributions

To address the problem of defining safety goals for subsystems in a composite system, this thesis makes the following contributions:

7.1.1 Formal definition of emergent and composable behaviors

I provided a formal definition of emergence within a framework of goal decomposition.

The main idea used in creation of this definition is that emergence exists as residual undefined behaviors in AND-reductions and OR-reductions of composite goals. This builds on prior work in goal elaboration that defined goal decomposition, but did not account for emergence in the missing subgoals of a partial decomposition. Emergence in AND-reductions indicates unknown or unrealizable behaviors that could cause the goal to be

violated, as defined in Equation 3.14. Emergence in OR- reductions indicates unknown or unrealizable behaviors that could cause the goal to be satisfied, as defined in Equation 3.23.

Chapter 3 presents the formal representation of a system decomposition and emergence. The scope of this representation is limited to system goals and subgoals defined within the temporal logic goal specification framework used by ICPA, which comes from Goal Oriented Requirements Engineering and the KAOS framework. It is also limited to describing a single decomposition of the system, not all possible decompositions. Formal definitions are presented for behaviors that are fully composable, emergent, and emergent but partially composable, with or without goal redundancy. In addition, the role of emergence in conjunctive and disjunctive goal forms is explained.

These definitions are generic mathematical representations that can be applied to any system requirements specification that can be defined by goals to be achieved at the system level and subgoals to be achieved by the subsystem. However, this means the formal definitions rely on the assumption that the system behaviors can be specified as expressions of temporal logic in a state transition system. This may limit its application in continuous systems, or systems with mixed continuous and discrete control.

For goal elaboration, particularly for system safety goals, formal representation of emergence within a composite system helps identify where emergence may occur in system decomposition, and how emergence may be handled. It also helps identify special cases in which an emergent behavior may be partially composable. Although the partial decomposition may not ensure the goal is always satisfied, it may identify behaviors that will definitely violate the goal, which is useful for safety. In this situation, it helps guide engineers in ‘best effort’ design.

7.1.2 Indirect Control Path Analysis (ICPA)

I created a technique for guiding system safety goal elaboration in a directed and documented way.

The main idea of ICPA is to combine familiar techniques from hazard analysis, such as the top-down approach of FTA and the table structure of FMEA, for system safety goal elaboration. Instead of tracing hazards back to potential faults, ICPA traces system safety goals to their indirect and direct control sources, guided by control flow of system state variables. The technique was applied to goals from an academic research system and a real system of significant complexity. Run-time monitoring of the subgoals in the latter system demonstrated that subgoals produced by ICPA partially compose the parent goals.

Chapter 4 proposed the ICPA technique for safety goal elaboration. The concepts of direct and indirect control were defined in Section 4.2, with respect to the KAOS goal elaboration framework. In addition, the ICPA format and procedure were explained in Sections 4.3 and 4.4, and illustrated with examples from a distributed elevator control system used in a graduate distributed embedded system course. ICPA uses a top-down analysis approach to trace state variables in a system safety goal to their indirect control sources in the system control architecture. Relationships among agents along the trace path and from agents to the parent state variable are defined. Goal coverage strategies, including a goal assignment and goal scope are used guide this process. The result of an ICPA is both a set of subsystem safety goals that satisfy the parent goal, and a record of the critical assumptions, goal realizability tactics, and goal coverage strategies used to define them. Section 4.5.3 also contained an explanation of controllability and observability requirements for goal realizability, including goal patterns for those requirements and for alternative goals.

In Chapters 5 and 6, ICPA was applied to a semi-autonomous automotive system developed by a commercial automotive research lab. The system safety goals and set of subgoals

produced by ICPA were then monitored in an implementation of the system in the CarSim® and Simulink® simulation environment. During monitoring, some goal violations were detected in the subgoal monitors, but some were not. This demonstrates that the subgoals produced in ICPA partially compose the parent goal, but not fully.

The techniques used in ICPA have been shown to be applicable to two types of distributed embedded systems, an elevator control system and an automotive system, and should be generally applicable to systems with the same type of closed loop control of physical actuation and sensing. ICPA relies on the assumption that the set of safety goals defined at the system level is complete and correct. That is, if the system safety goals are incorrect, the subgoals produced by ICPA will not ensure system safety. Another limitation of the technique is that in complex systems with many state variables or state variables with wide ranges of values, it is impossible to know if all indirect control relationships have been defined. In these systems, ICPA assists in best-effort design and implementation. The benefit of ICPA is that it presents a structured approach to identifying these indirect control sources and relationships among state variables. It also provides a format for documenting the design decisions used in defining the subgoals for the subsystems.

7.1.3 Hierarchical safety monitoring

I demonstrated that monitoring of system safety goals and subgoals can detect hazards at run-time.

Monitoring not only can indicate whether or not the subgoals produced with ICPA partially compose the parent goal, but also can identify key design defects that remain in the system at run-time. By building monitors for the automotive goals and subgoals to test ICPA, I also identified important design defects in the system under review, enumerated in Section 6.1.2.

Chapters 5 and 6 present the results of monitoring goals and subgoals produced by ICPA for a semi-autonomous automotive system. This was a real vehicle, though one intended for research and not as a commercial product. Run-time monitoring of the system safety goals and subgoals identified eleven design defects while the system was still under development. In some situations, subgoals were violated that did not result in parent goal violations, due to redundant goal coverage, or were so transient they may not be observable to the driver. Run-time monitoring allowed these design defects to be discovered and corrected when they otherwise might have been missed.

In addition, the process of defining the monitors revealed that some system safety goals cannot actually be monitored at the system level. In order to be monitored at the system level, the goal must constrain control of a state variable that cannot be sensed, or the goal must constrain control of state variables that are directly controlled by different subsystems. This would be true of any similar closed-loop control of physical actuation and sensing.

One limitation of the contribution was that run-time monitoring was only performed in one of the two systems used to illustrate the ICPA technique. Another limitation is that run-time monitoring required observability of all state variables in the goals and subgoals. In some system designs, this may require subgoal monitoring to be done locally in the subsystems. Despite these limitations, the system that was used for run-time monitoring was a real commercial automotive system, rather than an academic system designed for coursework. The issues uncovered with the system and information learned about ICPA during run-time monitoring reflect the issues that might be encountered in real safety-critical system development.

7.2 Future work

Evaluation of the ICPA raised the following issues that could be investigated in future research:

Physical sensing and actuation delay patterns. In the evaluation, any time acceleration changed, the jerk threshold was exceeded for at least one system state. This is characteristic of digital control, which uses sampled values rather than continuous values, and is reasonably easy to factor into temporal logic-based goals. However, changes in actuation have some residual delay in the resulting sensed values. Application of ICPA requires this delay to be accurately characterized and included in the goal elaboration process. It is likely that different types of actuation and sensing have characteristic delays. If so, perhaps these characteristics could be generalized into specific patterns for defining indirect control relationships in the ICPA, or perhaps a filtered value could be used instead.

Human factors of ICPA. One possible direction for future work would be to examine the human-factors issues involved with applying the ICPA. As mentioned in Section 5.4, the evaluation of ICPA in this thesis demonstrated that the technique can be applied to a real safety-critical distributed embedded system with closed-loop control of physical actuation and sensing to produce subgoals that partially compose the parent goal. This evaluation was done by the author of this thesis, and not the system engineers who designed, built and analyzed the system under review. As such, this evaluation does not show how well the ICPA can be applied by engineers in a real industrial development environment.

Human factors evaluation, in general, is difficult because it requires access to many people and many systems, preferably real developers working on real industrial products. For this thesis, it was extremely difficult to gain access to a single system of reasonably realistic complexity. In addition, the developers working on the system were too busy to participate

much in any of the steps used to perform this evaluation. As such, this future work direction would be extremely difficult without extensive participation by a specific organization.

Appendix A

Logic Operators, Acronyms, and Definitions

A.1. Temporal logic operators

$\mathbf{P}$	true in current state
$\neg\mathbf{P}$	false in current state
$\bullet\mathbf{P}$	true in previous state
$\blacklozenge\mathbf{P}$	true in some previous state
$\blacksquare\mathbf{P}$	true in all previous states
$\circ\mathbf{P}$	true in next state
$\lozenge\mathbf{P}$	true in current or some future state
$\square\mathbf{P}$	true in current and all future states
$\mathbf{P} \wedge \mathbf{Q}$	P *AND* Q
$\mathbf{P} \vee \mathbf{Q}$	P *OR* Q
$\mathbf{P} \rightarrow \mathbf{Q}$	P implies Q in current state
$\mathbf{P} \Rightarrow \mathbf{Q}$	$\square$(P $\rightarrow$ Q); P implies Q in all states
$\mathbf{P} \Leftrightarrow \mathbf{Q}$	P *iff* Q in all states
$\bullet\blacksquare_{<T}\mathbf{P}$	true for duration T in previous state
$\bullet\blacklozenge_{<T}\mathbf{P}$	true at least once in duration T in previous state
$@\mathbf{P}$	$\bullet\neg$P $\wedge$ P; true in current state, false in previous state
$\mathbf{S}_0 \models \mathbf{P}$	True in the initial state
$\Gamma \vdash \mathbf{P}$	P is syntactically derivable from the premises Γ
$\Gamma \nvdash \mathbf{P}$	P is not syntactically derivable from the premises Γ

A.2. Acronyms

ACC: Adaptive Cruise Control

CA: Collision Avoidance

FMEA: Failure Modes and Effects Analysis

FTA: Fault Tree Analysis

GORE: Goal Oriented Requirements Engineering

ICPA: Indirect Control Path Analysis

KAOS: Knowledge Acquisition in autOmated Specification

LCA: Lane Change Assist

PA: Park Assist

RCA: Rear Collision Avoidance

STPA Systems-Theoretic accident model and Process Analysis

A.3. Definitions

direct control: ability to change a state variable directly; direct control may be limited to one individual subsystem (e.g., one subsystem sends signals to an actuator) or may be shared among subsystems (e.g., multiple subsystems send the same type of request)

false negative: occurs when a goal violation is detected but no corresponding subgoal violations are detected

false positive: occurs when a subgoal violation is detected but no corresponding goal violation is detected

goal assignment: defines which indirect control sources have subgoals and how those subgoals relate to each other; categories include single responsibility, redundant responsibility, and shared responsibility.

goal coverage strategy: a plan for allocating subgoals to ensure that a high-level goal is met; it includes the goal assignment and goal scope.

goal scope: defines how closely the safety subgoals meet the system safety goal; categories include nonrestrictive and restrictive.

hit: occurs when a goal violation is detected and a corresponding goal violation is detected

indirect control: ability to influence change in a state variable; sources include hardware actuation, system dynamics, and environmental agents that change sensed state variables, or inputs from other subsystems

Appendix B

Goal Realizability Patterns and Alternative Goals

Table B.1. Goal realizability patterns and alternative goals for $A \Rightarrow B$, $\bigcirc A \Rightarrow B$, and $A \Rightarrow \bigcirc B$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A $\Rightarrow$ B	A		$\Box \neg$ A	Yes
	A	B	$\Box \neg$ A	Yes
	B		$\Box$ B	Yes
	B	A	$\Box$ B	Yes
	A, B			No
$\bigcirc$ A $\Rightarrow$ B	A		$\Box \neg$ A	Yes
	A	B	$\Box \neg$ A	Yes
	B		$\Box$ B	Yes
	B	A		No
	A, B			No
A $\Rightarrow \bigcirc$ B	A		$\Box \neg$ A	Yes
	A	B	$\bigcirc \neg$ B $\Rightarrow \neg$ A	No
	B		$\Box$ B	Yes
	B	A	$\Box$ B	Yes
	A, B		$\bigcirc \neg$ B $\Rightarrow \neg$ A	No

Table B.2. Goal realizability patterns and alternative goals for $A \vee B \Rightarrow C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A ∨ B ⇒ C	A		Not realizable with or without restriction	
	A	B	Not realizable with or without restriction	
	A	C	Not realizable with or without restriction	
	A	B, C	Not realizable with or without restriction	
	B		Not realizable with or without restriction	
	B	A	Not realizable with or without restriction	
	B	C	Not realizable with or without restriction	
	B	A, C	Not realizable with or without restriction	
	C		❑ C	Yes
	C	A	❑ C	Yes
	C	B	❑ C	Yes
	C	A, B	❑ C	Yes
	A, B		❑ ¬ (A ∨ B)	Yes
	A, B	C	❑ ¬ (A ∨ B)	Yes
	A, C		❑ C	Yes
	A, C	B	❑ C	Yes
	B, C		❑ C	Yes
	B, C	A	❑ C	Yes
	A, B, C			No

Table B.3. Goal realizability patterns and alternative goals for ●$A \vee B \Rightarrow C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
● A ∨ B ⇒ C	A		Not realizable with or without restriction	
	A	B	Not realizable with or without restriction	
	A	C	Not realizable with or without restriction	
	A	B, C	Not realizable with or without restriction	
	B		Not realizable with or without restriction	
	B	A	Not realizable with or without restriction	
	B	C	Not realizable with or without restriction	
	B	A, C	Not realizable with or without restriction	
	C		❑ C	Yes
	C	A	❑ C	Yes
	C	B	❑ C	Yes
	C	A, B	❑ C	Yes
	A, B		❑ ¬ (A ∨ B)	Yes
	A, B	C	❑ ¬ (A ∨ B)	Yes
	A, C		❑ C	Yes
	A, C	B	❑ C	Yes
	B, C		❑ C	Yes
	B, C	A		No
	A, B, C			No

Table B.4. Goal realizability patterns and alternative goals for $A \vee B \Rightarrow \bullet C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A ∨ B ⇒ ● C	A		Not realizable with or without restriction	
	A	B	Not realizable with or without restriction	
	A	C	Not realizable with or without restriction	
	A	B, C	Not realizable with or without restriction	
	B		Not realizable with or without restriction	
	B	A	Not realizable with or without restriction	
	B	C	Not realizable with or without restriction	
	B	A, C	Not realizable with or without restriction	
	C		❑ C	Yes
	C	A	❑ C	Yes
	C	B	❑ C	Yes
	C	A, B	❑ C	Yes
	A, B		❑ ¬ (A ∨ B)	Yes
	A, B	C	● ¬ C ⇒ ¬ (A ∨ B)	No
	A, C		❑ C	Yes
	A, C	B	❑ C	Yes
	B, C		❑ C	Yes
	B, C	A	❑ C	Yes
	A, B, C		● ¬ C ⇒ ¬ (A ∨ B)	No

Table B.5. Goal realizability patterns and alternative goals for $A \wedge B \Rightarrow C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A $\wedge$ B $\Rightarrow$ C	A		❑ $\neg$ A	Yes
	A	B	❑ $\neg$ A	Yes
	A	C	❑ $\neg$ A	Yes
	A	B,C	❑ $\neg$ A	Yes
	B		❑ $\neg$ B	Yes
	B	A	❑ $\neg$ B	Yes
	B	C	❑ $\neg$ B	Yes
	B	A,C	❑ $\neg$ B	Yes
	C		❑ C	Yes
	C	A	❑ C	Yes
	C	B	❑ C	Yes
	C	A, B	❑ C	Yes
	A, B		❑ $\neg$ (A $\wedge$ B)	Yes
	A, B	C	❑ $\neg$ (A $\wedge$ B)	Yes
	A, C		A $\Rightarrow$ C	Yes
	A, C	B	A $\Rightarrow$ C	Yes
	B, C		B $\Rightarrow$ C	Yes
	B, C	A	B $\Rightarrow$ C	Yes
	A, B, C			No

Table B.6. Goal realizability patterns and alternative goals for $\bullet A \wedge B \Rightarrow C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A ∧ B ⇒ ● C	A		❑ ¬ A	Yes
	A	B	❑ ¬ A	Yes
	A	C	❑ ¬ A	Yes
	A	B,C	❑ ¬ A	Yes
	B		❑ ¬ B	Yes
	B	A	❑ ¬ B	Yes
	B	C	❑ ¬ B	Yes
	B	A,C	❑ ¬ B	Yes
	C		❑ C	Yes
	C	A	❑ C	Yes
	C	B	❑ C	Yes
	C	A, B	❑ C	Yes
	A, B		❑ ¬ (A ∧ B)	Yes
	A, B	C	● ¬ C ⇒ ¬ (A ∧ B)	No
	A, C		A ⇒ C	Yes
	A, C	B	A ⇒ C	Yes
	B, C		B ⇒ C	Yes
	B, C	A	B ⇒ C	Yes
	A, B, C			No

Table B.7. Goal realizability patterns and alternative goals for $A \wedge B \Rightarrow \bullet C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
$\bullet A \wedge B \Rightarrow C$	A		$\square \neg A$	Yes
	A	B	$\square \neg A$	Yes
	A	C	$\square \neg A$	Yes
	A	B,C	$\square \neg A$	Yes
	B		$\square \neg B$	Yes
	B	A	$\bullet A \Rightarrow \neg B$	Yes
	B	C	$\square \neg B$	Yes
	B	A,C	$\bullet A \Rightarrow \neg B$	Yes
	C		$\square C$	Yes
	C	A	$\bullet A \Rightarrow C$	Yes
	C	B	$\square C$	Yes
	C	A, B	$\bullet A \Rightarrow C$	Yes
	A, B		$\square \neg (\bullet A \wedge B)$	Yes
	A, B	C	$\square \neg (\bullet A \wedge B)$	Yes
	A, C		$\bullet A \Rightarrow C$	Yes
	A, C	B	$\bullet A \Rightarrow C$	Yes
	B, C		$B \Rightarrow C$	Yes
	B, C	A		No
	A, B, C			No

Table B.8. Goal realizability patterns and alternative goals for $A \Rightarrow B \wedge C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A $\Rightarrow$ B $\wedge$ C	A		❑ $\neg$ A	Yes
	A	B	❑ $\neg$ A	Yes
	A	C	❑ $\neg$ A	Yes
	A	B,C	❑ $\neg$ A	Yes
	B		Not realizable with or without restriction	
	B	A	Not realizable with or without restriction	
	B	C	Not realizable with or without restriction	
	B	A,C	Not realizable with or without restriction	
	C		Not realizable with or without restriction	
	C	A	Not realizable with or without restriction	
	C	B	Not realizable with or without restriction	
	C	A,C	Not realizable with or without restriction	
	A, B		❑ $\neg$ A	Yes
	A, B	C	❑ $\neg$ A	Yes
	A, C		❑ $\neg$ A	Yes
	A, C	B	❑ $\neg$ A	Yes
	B, C		❑ (B $\wedge$ C)	Yes
	B, C	A	❑ (B $\wedge$ C)	Yes
	A, B, C			No

Table B.9. Goal realizability patterns and alternative goals for ●$A \Rightarrow B \wedge C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
● A ⇒ B ∧ C	A		❑ ¬ A	Yes
	A	B	❑ ¬ A	Yes
	A	C	❑ ¬ A	Yes
	A	B,C	❑ ¬ A	Yes
	B		Not realizable with or without restriction	
	B	A	Not realizable with or without restriction	
	B	C	Not realizable with or without restriction	
	B	A,C	Not realizable with or without restriction	
	C		Not realizable with or without restriction	
	C	A	Not realizable with or without restriction	
	C	B	Not realizable with or without restriction	
	C	A,C	Not realizable with or without restriction	
	A, B		❑ ¬ A	Yes
	A, B	C	❑ ¬ A	Yes
	A, C		❑ ¬ A	Yes
	A, C	B	❑ ¬ A	Yes
	B, C		❑ (B ∧ C)	Yes
	B, C	A		No
	A, B, C			No

Table B.10. Goal realizability patterns and alternative goals for $A \Rightarrow \bullet B \wedge C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A $\Rightarrow$ ● B $\wedge$ C	A		❑ $\neg$ A	Yes
	A	B	❑ $\neg$ A	Yes
	A	C	❑ $\neg$ A	Yes
	A	B,C	❑ $\neg$ A	Yes
	B		Not realizable with or without restriction	
	B	A	Not realizable with or without restriction	
	B	C	Not realizable with or without restriction	
	B	A,C	Not realizable with or without restriction	
	C		Not realizable with or without restriction	
	C	A	Not realizable with or without restriction	
	C	B	Not realizable with or without restriction	
	C	A,C	Not realizable with or without restriction	
	A, B		❑ $\neg$ A	Yes
	A, B	C	❑ $\neg$ A	Yes
	A, C		❑ $\neg$ A	Yes
	A, C	B	❑ $\neg$ A	Yes
	B, C		❑ (● B $\wedge$ C)	Yes
	B, C	A	❑ (● B $\wedge$ C)	No
	A, B, C			No

Table B.11. Goal realizability patterns and alternative goals for $A \Rightarrow B \vee C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
A $\Rightarrow$ B $\vee$ C	A		❑ $\neg$ A	Yes
	A	B	❑ $\neg$ A	Yes
	A	C	❑ $\neg$ A	Yes
	A	B,C	❑ $\neg$ A	Yes
	B		❑ B	Yes
	B	A	❑ B	Yes
	B	C	❑ B	Yes
	B	A,C	❑ B	Yes
	C		❑ C	Yes
	C	A	❑ C	Yes
	C	B	❑ C	Yes
	C	A, B	❑ C	Yes
	A, B		A $\Rightarrow$ B	Yes
	A, B	C	A $\Rightarrow$ B	Yes
	A, C		A $\Rightarrow$ C	Yes
	A, C	B	A $\Rightarrow$ C	Yes
	B, C		❑ (B $\vee$ C)	Yes
	B, C	A	❑ (B $\vee$ C)	Yes
	A, B, C			No

Table B.12. Goal realizability patterns and alternative goals for ● $A \Rightarrow B \vee C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
● A ⇒ B ∨ C	A		❑ ¬ A	Yes
	A	B	❑ ¬ A	Yes
	A	C	❑ ¬ A	Yes
	A	B,C	❑ ¬ A	Yes
	B		❑ B	Yes
	B	A	● A ⇒ B	Yes
	B	C	❑ B	Yes
	B	A, C	● A ⇒ B	Yes
	C		❑ C	Yes
	C	A	● A ⇒ C	Yes
	C	B	❑ C	Yes
	C	A, B	● A ⇒ C	Yes
	A, B		● A ⇒ B	Yes
	A, B	C	● A ⇒ B	Yes
	A, C		● A ⇒ C	Yes
	A, C	B	● A ⇒ C	Yes
	B, C		❑ (B ∨ C)	Yes
	B, C	A		No
	A, B, C			No

Table B.13. Goal realizability patterns and alternative goals for $A \Rightarrow \bullet B \vee C$

Pattern	Controllable	Observable	Alternative Goal	Restrictive
$A \Rightarrow \bullet B \vee C$	A		$\square \neg A$	Yes
	A	B	$\bullet \neg B \Rightarrow \neg A$	Yes
	A	C	$\square \neg A$	Yes
	A	B,C	$\bullet \neg B \Rightarrow \neg A$	Yes
	B		$\square B$	Yes
	B	A	$\square B$	Yes
	B	C	$\square B$	Yes
	B	A,C	$\square B$	Yes
	C		$\square C$	Yes
	C	A	$\square C$	Yes
	C	B	$\bullet \neg B \Rightarrow C$	Yes
	C	A, B	$\bullet \neg B \Rightarrow C$	Yes
	A, B		$\bullet \neg B \Rightarrow \neg A$	Yes
	A, B	C	$\bullet \neg B \Rightarrow \neg A$	Yes
	A, C		$A \Rightarrow C$	Yes
	A, C	B	$A \wedge \bullet \neg B \Rightarrow C$	No
	B, C		$\bullet \neg B \Rightarrow C$	Yes
	B, C	A	$\bullet \neg B \Rightarrow C$	Yes
	A, B, C			No

Appendix C

ICPA for a Semi-autonomous Automotive System

ICPA for Achieve[AutoAccelBelowThreshold] (1 of 4)

System Safety Goal

Goal: Achieve[AutoAccelBelowThreshold]
InformalDef: *Vehicle acceleration caused by autonomous vehicle control shall not exceed 2 m/s^2.*
FormalDef: $\forall$ va: VehicleAcceleration
IsSubsystem(va.source) $\Rightarrow$ va.value $\leq$ 2 m/s^2

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	Arbiter	ac: AccelerationCommand	❑ (va.value = ● ac.value) % Acceleration is equal to the previous acceleration command value	01
			❑ (va.source = ● ac.source) % The source of acceleration is equal to the source of the previous acceleration command	02
			❑ (ac.source $\in$ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	03

Figure C.1. ICPA for Achieve[AutoAccelBelowThreshold] (1 of 4)

ICPA for Achieve[AutoAccelBelowThreshold] (2 of 4)				
va	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	04
			IsSubsystem(ac.source) ⇔ (∃ ar: ac.source = ● ar.source) % If the source of the acceleration command is a subsystem, then % there exists an acceleration request such that the source of the % acceleration command is the source of that acceleration request, % and vice versa	05
			❑ IsSubsystem(ar.source) % The source of the acceleration request is a subsystem	06
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	07

Figure C.2. ICPA for Achieve[AutoAccelBelowThreshold] (2 of 4)

ICPA for Achieve[AutoAccelBelowThreshold] (3 of 4)	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter; Secondary: CA, RCA, ACC, LCA, PA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Also, OR reduction is used for CA, RCA, ACC, LCA, and PA goals.)
Goal Elaboration	
IsSubsystem(va.source) $\Rightarrow$ va.value $\leq$ 2 m/s^2	Indirect Control Relationships
(● IsSubsystem(ac.source)) $\Rightarrow$ ● (ac.value $\leq$ 2 m/s^2)	01, 02 – Introduce accuracy/actuation goal tactic Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02)
(ac.source = ● ar.source) $\Rightarrow$ (ac.value $\leq$ 2 m/s^2)	05 – Introduce accuracy/actuation goal tactic
(ac.source = ● ar.source) $\Rightarrow$ ● (ar.value $\leq$ 2 m/s^2)	04 – Introduce accuracy/actuation goal tactic
❑(ar.value $\leq$ 2 m/s^2)	OR-Reduction

Figure C.3. ICPA for Achieve[AutoAccelBelowThreshold] (3 of 4)

ICPA for Achieve[AutoAccelBelowThreshold] (4 of 4)
Subsystem Safety Goals
Subsystem: Arbiter **Controls:** AcelerationCommand **Observes:** AccelerationRequest **Goal:** Achieve[AutoAccelCommandBelowThreshold] **InformalDef:** *Vehicle acceleration commands caused by autonomous vehicle control shall not exceed 2 m/s^2.* **FormalDef:** $\forall$ ac: AccelerationCommand, ar: AccelerationRequest (ac.source = ● ar.source) $\Rightarrow$ (ac.value $\leq$ 2 m/s^2)
Subsystem: CA, RCA, ACC, LCA, PA **Controls:** AcelerationRequest **Observes:** **Goal:** Maintain[AutoAccelRequestBelowThreshold] **InformalDef:** *Vehicle acceleration requests caused by autonomous vehicle control shall not exceed 2 m/s^2.* **FormalDef:** $\forall$ ar: AccelerationRequest ❑(ar.value $\leq$ 2 m/s^2)

Figure C.4. ICPA for Achieve[AutoAccelBelowThreshold] (4 of 4)

ICPA for Achieve[AutoJerkBelowThreshold] (1 of 4)

System Safety Goal

Goal: Achieve[AutoJerkBelowThreshold]
InformalDef: *Vehicle jerk caused by autonomous vehicle control shall not exceed 2.5 m/s^3.*
FormalDef: $\forall$ vj: VehicleJerk
IsSubsystem(vj.source) $\Rightarrow$ vj.value $\leq$ 2.5 m/s^3

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
vj	Arbiter	ac: AccelerationCommand	❑ (vj.value = VehicleJerkResponse(● ac.value, ● ● ac.value)) % Jerk is equal to the jerk response caused by the values of the % previous two acceleration commands	01
			❑ (vj.source = ● ac.source) % The source of jerk is the source of the previous acceleration % command	02
			❑ (ac.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	03

Figure C.5. ICPA for Achieve[AutoJerkBelowThreshold] (1 of 4)

ICPA for Achieve[AutoJerkBelowThreshold] (2 of 4)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
vj	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	04
			IsSubsystem(ac.source) ⇔ (∃ ar: ac.source = ● ar.source) % If the source of the acceleration command is a subsystem, then % there exists an acceleration request such that the source of the % acceleration command is the source of that acceleration request, % and vice versa	05
			❑ IsSubsystem(ar.source) % The source of the acceleration request is a subsystem	06
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	07

Figure C.6. ICPA for Achieve[AutoJerkBelowThreshold] (2 of 4)

ICPA for Achieve[AutoJerkBelowThreshold] (3 of 4)	
Goal Elaboration	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter; Secondary: CA, RCA, ACC, LCA, PA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Also, OR reduction is used for CA, RCA, ACC, LCA, and PA goals)
Goal Elaboration	
IsSubsystem(vj.source) $\Rightarrow$ vj.value $\leq$ 2.5 m/s^3	Indirect Control Relationships
(● IsSubsystem(ac.source)) $\Rightarrow$ (VehicleJerkResponse(● ac.value, ● ● ac.value) $\leq$ 2.5 m/s^3)	01, 02 – Introduce accuracy/actuation goal tactic Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02)
(ac.source = ● ar.source) $\Rightarrow$ (VehicleJerkResponse(ac.value, ● ac.value) $\leq$ 2.5 m/s^3)	05 – Introduce accuracy/actuation goal tactic
((ac.source = ● ar.source) $\Rightarrow$ (VehicleJerkResponse(● ar.value, ● ● ar.value) $\leq$ 2.5 m/s^3)	04 – Introduce accuracy/actuation goal tactic
❑(VehicleJerkResponse(ar.value, ● ar.value) $\leq$ 2.5 m/s^3)	OR-Reduction

Figure C.7. ICPA for Achieve[AutoJerkBelowThreshold] (3 of 4)

ICPA for Achieve[AutoJerkBelowThreshold] (4 of 4)
Subsystem Safety Goals
Subsystem: Arbiter **Controls:** AcelerationCommand **Observes:** AccelerationRequest **Goal:** Achieve[AutoAccelCommandJerkBelowThreshold] **InformalDef:** *Vehicle acceleration commands caused by autonomous vehicle control shall not produce jerk that exceeds 2.5 m/s^2.* **FormalDef:** $\forall$ ac: AccelerationCommand, ar: AccelerationRequest ((ac.source = $\bullet$ ar.source) $\Rightarrow$ (VehicleJerkResponse($\bullet$ ar.value, $\bullet\bullet$ ar.value) $\leq$ 2.5 m/s^3)
Subsystem: CA, RCA, ACC, LCA, PA **Controls:** AcelerationRequest **Observes:** **Goal:** Maintain[AutoAccelRequestJerkBelowThreshold] **InformalDef:** *Vehicle acceleration requests caused by autonomous vehicle control shall not produce jerk that exceeds 2.5 m/s^2.* **FormalDef:** $\forall$ ar: AccelerationRequest $\square$(VehicleJerkResponse(ar.value, $\bullet$ ar.value) $\leq$ 2.5 m/s^3)

Figure C.8. ICPA for Achieve[AutoJerkBelowThreshold] (4 of 4)

ICPA for Achieve[SubsystemAccelSteeringAgreement] (1 of 5)

System Safety Goal

Goal: Achieve[SubsystemAccelSteeringAgreement]

InformalDef: *If a subsystem a) requests control of acceleration and steering and b) is granted control of either acceleration or steering, then the subsystem shall control both acceleration and steering.*

FormalDef: $\forall$ va: VehicleAcceleration, vst: VehicleSteering, sn: SubsystemName

$\bullet$ RequestingAcceleration(sn) $\wedge$ $\bullet$ RequestingSteering(sn) $\wedge$ ((va.source = sn) $\vee$ (vst.source = sn))

$\Rightarrow$ (va.source = vst.source = sn)

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
sn	Arbiter		$\square$ (sn $\in$ {Arbiter, CA, RCA, ACC, LCA, PA}) % A subsystem is either the arbiter or one of the feature subsystems	01
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	((ar.source = sn) $\wedge$ ar.active) $\Leftrightarrow$ RequestingAcceleration(sn) % If an acceleration request is active, then the source of that % acceleration request is requesting acceleration, and vice versa	02
		sr: AccelerationRequest	((sr.source = sn) $\wedge$ sr.active) $\Leftrightarrow$ RequestingSteering(sn) % If a steering request is active, then the source of that steering % request is requesting steering, and vice versa	03

Figure C.9. ICPA for Achieve[SubsystemAccelSteeringAgreement] (1 of 5)

ICPA for Achieve[SubsystemAccelSteeringAgreement] (2 of 5)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	Arbiter	ac: AccelerationCommand	❑ (va.value = ● ac.value) % Acceleration is equal to the previous acceleration command value	04
			❑ (va.source = ● ac.source) % The source of acceleration is equal to the source of the previous acceleration command	05
			❑ (ac.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	06
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	07
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	08
			● ¬ ar.active ⇒ ¬ (ac.source = ● ar.source) % If a previous acceleration request is not active, then the source of % an acceleration command is not the source of that previous % acceleration request	09

Figure C.10. ICPA for Achieve[SubsystemAccelSteeringAgreement] (2 of 5)

ICPA for Achieve[SubsystemAccelSteeringAgreement] (3 of 5)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
vst	Arbiter	sc: SteeringCommand	❑ (vst.value = ● sc.value) % The value of steering is equal to the value of the previous steering % command	10
			❑ (vst.source = ● sc.source) % The source of steering is the source of the previous steering % command	11
			❑ (sc.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of a steering command is either the driver or one of the % feature subsystems	12
	CA, RCA, ACC, LCA, PA	sr: SteeringRequest	sc.source = ● sr.source ⇒ sc.value = ● sr.value % If the source of a steering command is the source of a previous % steering request, then the value of the steering command is equal % to the value of the previous steering request	13
			❑ (sr.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of a steering request is one of the feature subsystems	14
			● ¬ sr.active ⇒ ¬ (sc.source = ● sr.source) % If a previous steering request is not active, then the source of a % steering command is not the source of that previous steering % request	15

Figure C.11. ICPA for Achieve[SubsystemAccelSteeringAgreement] (3 of 5)

ICPA for Achieve[SubsystemAccelSteeringAgreement] (4 of 5)	
Goal Coverage Strategy	
Goal Assignment	Single Responsibility (Arbiter)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Also, scope of the goal is expanded by incorporating critical assumption.)
Goal Elaboration	
(●RequestingAcceleration(sn) ∧ ●RequestingSteering(sn) ∧((va.source = sn) ∨ (vst.source = sn))) ⇒ (va.source = vst.source = sn)	Indirect Control Relationships
(●RequestingAcceleration(sn) ∧ ●RequestingSteering(sn) ∧((ac.source = sn) ∨ (sc.source = sn))) ⇒ (ac.source = sc.source = sn)	05, 11 – Introduce accuracy/actuation goal tactic
(●((ar.source = sn) ∧ ar.active) ∧ ●((sr.source = sn) ∧ sr.active) ∧((ac.source = ●ar.source) ∨ (sc.source = ●sr.source))) ⇒ (ac.source = sc.source = ●ar.source = ●sr.source = sn)	02, 03 – Introduce accuracy/actuation goal tactic
(●((ar.source = sn) ∧ ar.active) ∧ ●((sr.source = sn) ∧ sr.active) ∧((ac.source = ●ar.source) ∨ (sc.source = ●sr.source))) ⇒ ((ac.source = sc.source = ●ar.source = ●sr.source = sn) ∧ (ac.value = sc.value = ●ar.value = ●sr.value = sn))	07, 13 – Expanding scope by incorporating critical assumption into the subgoal

Figure C.12. ICPA for Achieve[SubsystemAccelSteeringAgreement] (4 of 5)

ICPA for Achieve[SubsystemAccelSteeringAgreement] (5 of 5)

Subsystem Safety Goals

Subsystem: Arbiter
Controls: AcelerationCommand, SteeringCommand
Observes: AccelerationRequest, SteeringRequest
Goal: Achieve[AccelSteeringCommandAgreement]
InformalDef: *If subsystem a) requests control of both the acceleration command and the steering command and b) is granted control of either the acceleration command or the steering command, then the subsystem shall be granted control of both the acceleration command and the steering command.*
FormalDef: $\forall$ ac: AccelerationCommand, ar: AccelerationRequest, sc: SteeringCommand, sr: SteeringRequest

$$(\bullet((ar.source = sn) \wedge ar.active) \wedge \bullet((sr.source = sn) \wedge sr.active) \wedge ((ac.source = \bullet ar.source) \vee (sc.source = \bullet sr.source))) \Rightarrow ((ac.source = sc.source = \bullet ar.source = \bullet sr.source = sn) \wedge (ac.value = sc.value = \bullet ar.value = \bullet sr.value = sn))$$

Figure C.13. ICPA for Achieve[SubsystemAccelSteeringAgreement] (5 of 5)

ICPA for Achieve[NoAutoAccelFromStop] (1 of 4)
System Safety Goal
Goal: Achieve[NoAutoAccelFromStop] **InformalDef:** *If a) the vehicle is stopped for a duration of StoppedTime and* *b) the throttle pedal has not been applied within the preceding GoTime and* *c) a subsystem is controlling acceleration and* *d) the HMI has not sent a go signal to the controlling subsystem within the preceeding AccelerationTime,* *then there shall be no vehicle acceleration* **FormalDef:** $\forall$ va: VehicleAcceleration, vsp: VehicleSpeed, sn: SubsystemName, hmi: HumanMachineInterface, tp: ThrottlePedal st: StoppedTime, gt: GoTime $(\bullet\blacksquare_{<st}$ IsStopped(vsp.value) $\wedge \bullet\blacksquare_{<gt} \neg$ @IsApplied(tp) $\wedge$ (va.source = sn) $\wedge \bullet\blacksquare_{<gt} \neg$ @Go(hmi, sn)) $\Rightarrow \neg$ IsAccelerating(va.value)

Figure C.14. ICPA for Achieve[NoAutoAccelFromStop] (1 of 4)

ICPA for Achieve[NoAutoAccelFromStop] (2 of 4)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	Arbiter	ac: AccelerationCommand	❑ (va.value = ● ac.value) % Acceleration is equal to the previous acceleration command value	01
			❑ (va.source = ● ac.source) % The source of acceleration is equal to the source of the previous acceleration command	02
			❑ (ac.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	03
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	04
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	05
sn	Arbiter		❑ (sn ∈ {Arbiter, CA, RCA, ACC, LCA, PA}) % A subsystem is either the arbiter or one of the feature subsystems	06
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	❑ (ar.source = sn) % The source of an acceleration request is a subsystem	07

Figure C.15. ICPA for Achieve[NoAutoAccelFromStop] (2 of 4)

ICPA for Achieve[NoAutoAccelFromStop] (3 of 4)	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter, Secondary: CA, RCA, ACC, LCA, PA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command, worst-case delays between features & arbiter.)
Goal Elaboration	
$(\bullet\blacksquare_{<st}$ IsStopped(vsp.value) $\wedge \neg (\bullet\blacksquare_{<gt}$ @IsApplied(tp)) $\wedge$ (va.source = sn) $\wedge \neg (\bullet\blacksquare_{<gt}$ @Go(hmi, sn))) $\Rightarrow \neg$ IsAccelerating(va.value)	Indirect Control Relationships
$(\bullet\blacksquare_{<st-1}$ IsStopped(vsp.value) $\wedge (\bullet\blacksquare_{<gt-1} \neg$ @IsApplied(tp)) $\wedge$ (ac.source = sn) $\wedge (\bullet\blacksquare_{<gt-1}$ @Go(hmi, sn))) $\Rightarrow \neg$ IsAccelerating(ac.value)	01, 02 – Introduce accuracy/actuation goal tactic Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02}
$(\bullet\blacksquare_{<st-2}$ IsStopped(vsp.value) $\wedge (\bullet\blacksquare_{<gt-2} \neg$ @IsApplied(tp)) $\wedge (\bullet\blacksquare_{<gt-2} \neg$ @Go(hmi, sn))) $\Rightarrow \neg$ IsAccelerating(ar.value)	04 – Introduce accuracy/actuation goal tactic

Figure C.16. ICPA for Achieve[NoAutoAccelFromStop] (3 of 4)

ICPA for Achieve[NoAutoAccelFromStop] (4 of 4)

Subsystem Safety Goals

Subsystem: Arbiter
Controls: AcelerationCommand
Observes: VehicleSpeed, SubsystemName, ThrottlePedal, HumanMachineInterface, StoppedTime, GoTime
Goal: Achieve[AutoAccelCommandBelowThreshold]
InformalDef: *If a) the vehicle is stopped for a duration of StoppedTime and*
b) the throttle pedal has not been applied within the preceding GoTime and
c) a subsystem is controlling acceleration and
d) the HMI has not sent a go signal to the controlling subsystem within the preceeding GoTime,
then the vehicle acceleration command shall be set to
FormalDef: ∀ ac: AccelerationCommand, vsp: VehicleSpeed, sn: SubsystemName, hmi: HumanMachineInterface, tp:ThrottlePedal, st: StoppedTime, gt: GoTime
$(\bullet\blacksquare_{<st\text{-}1}$ IsStopped(vsp.value) $\wedge$ $(\bullet\blacksquare_{<gt\text{-}1} \neg$ @IsApplied(tp)) $\wedge$ (ac.source = sn) $\wedge$ $(\bullet\blacksquare_{<gt\text{-}1} \neg$ @Go(hmi, sn)))
$\Rightarrow \neg$ IsAccelerating(ac.value)

Subsystem: CA, RCA, ACC, LCA, PA
Controls: AcelerationRequest
Observes: VehicleSpeed, SubsystemName, ThrottlePedal, HumanMachineInterface, StoppedTime, GoTime
Goal: Achieve[AutoAccelRequestBelowThreshold]
InformalDef: *If a) the vehicle is stopped for a duration of StoppedTime and*
b) the throttle pedal has not been applied within the preceding GoTime and
c) a subsystem is controlling acceleration and
d) the HMI has not sent a go signal to the controlling subsystem within the preceeding GoTime,
then the vehicle acceleration command shall be set to
FormalDef: ∀ ar: AccelerationRequest, vsp: VehicleSpeed, sn: SubsystemName, hmi: HumanMachineInterface, tp:ThrottlePedal, st: StoppedTime, gt: GoTime
$(\bullet\blacksquare_{<st\text{-}2}$ IsStopped(vsp.value) $\wedge$ $(\bullet\blacksquare_{<gt\text{-}2} \neg$ @IsApplied(tp)) $\wedge$ $(\bullet\blacksquare_{<gt\text{-}2} \neg$ @Go(hmi, sn)))
$\Rightarrow \neg$ IsAccelerating(ar.value)

Figure C.17. ICPA for Achieve[NoAutoAccelFromStop] (4 of 4)

ICPA for Achieve[DriverForwardAccelOverride] (1 of 4)

System Safety Goal

Goal: Achieve[DriverForwardAccelOverride]

InformalDef: *If a) the vehicle is moving in the forward direction and*
b) the driver is applying the brake pedal or the throttle pedal and
c) a subsystem is requesting a vehicle acceleration greater than or equal to -2 m/s^2 (i.e., not requesting a "hard" stop of the vehicle),
then the subsystem shall not control vehicle acceleration.

FormalDef: $\forall$ va: VehicleAcceleration, sn: SubsystemName, bp: BrakePedal, tp: ThrottlePedal
$\bullet\bullet$ (InForwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active) $\wedge$ RequestingAcceleration(sn)
$\wedge$ (RequestedAcceleration(sn) $\geq$ -2 m/s^2)) $\Rightarrow$ $\neg$ (va.source = sn)

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	Arbiter	ac: AccelerationCommand	❑ (va.value = $\bullet$ ac.value) % Acceleration is equal to the previous acceleration command value	01
			❑ (va.source = $\bullet$ ac.source) % The source of acceleration is equal to the source of the previous acceleration command	02
			❑ (ac.source $\in$ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	03

Figure C.18. ICPA for Achieve[DriverForwardAccelOverride] (1 of 4)

ICPA for Achieve[DriverForwardAccelOverride] (2 of 4)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	04
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	05
			● ¬ ar.active ⇒ ¬ (ac.source = ● ar.source) % If a previous acceleration request is not active, then the source of % an acceleration command is not the source of that previous % acceleration request	06
sn	Arbiter		❑ (sn ∈ {Arbiter, CA, RCA, ACC, LCA, PA}) % A subsystem is either the arbiter or one of the feature subsystems	07
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	((ar.source = sn) ∧ ar.active) ⇔ RequestingAcceleration(sn) % If an acceleration request is active, then the source of that % acceleration request is requesting acceleration, and vice versa	08
			((ar.source = sn) ∧ ar.active) ⇔ (RequestedAcceleration(sn) = ar.value) % If an acceleration request is active, then the requested acceleration % of the source of the acceleration request is the value of the % acceleration request.	09

Figure C.19. ICPA for Achieve[DriverForwardAccelOverride] (2 of 4)

ICPA for Achieve[DriverForwardAccelOverride] (3 of 4)	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter, Secondary: CA, RCA, ACC, LCA, PA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Also, OR reduction is used for CA, RCA, ACC, LCA, and PA goals, and restricts acceleration requests one state earlier than necessary.)
Goal Elaboration	
●● (InForwardMotion(vsp.value) ∧ (bp.active ∨ tp.active) ∧ RequestingAcceleration(sn) ∧ (RequestedAcceleration(sn) ≥ -2.5 m/s^2)) ⇒ ¬ (va.source = sn)	Indirect Control Relationships
● (InForwardMotion(vsp.value) ∧ (bp.active ∨ tp.active) ∧ ((ar.source = sn) ∧ ar.active) ∧ (ar.value ≥ -2.5 m/s^2)) ⇒ ¬ (ac.source = ● ar.source)	01, 02, 08, 09 – Introduce accuracy/actuation goal tactic. Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02)
● (InForwardMotion(vsp.value) ∧ (bp.active ∨ tp.active)) ⇒ (¬ ar.active ∨ (ar.value < -2 m/s^2))	OR reduction. Also restricts acceleration requests one state earlier than necessary.

Figure C.20. ICPA for Achieve[DriverForwardAccelOverride] (3 of 4)

ICPA for Achieve[DriverForwardAccelOverride] (4 of 4)
Subsystem Safety Goals
Subsystem: Arbiter **Controls:** AcelerationCommand **Observes:** AccelerationRequest, BrakePedal, ThrottlePedal **Goal:** Acheive[DriverForwardAccelOverrideAccelCommand] **InformalDef:** *If a) the vehicle is moving in the forward direction and* *b) the driver is applying the brake pedal or the throttle pedal and* *c) a subsystem is requesting a vehicle acceleration greater than or equal to -2 m/s^2 (i.e., not requesting a "hard" stop of the vehicle),* *then the source of the acceleration command shall not be a subsystem* **FormalDef:** $\forall$ ac: AccelerationCommand, bp: BrakePedal, ar:AccelerationRequest, tp:ThrottlePedal $\bullet$ (InForwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active) $\wedge$ ((ar.source = sn) $\wedge$ ar.active) $\wedge$ (ar.value $\geq$ -2 m/s^2)) $\Rightarrow \neg$ (ac.source = $\bullet$ ar.source)
Subsystem: CA, RCA, ACC, LCA, PA **Controls:** AcelerationRequest **Observes:** BrakePedal, ThrottlePedal **Goal:** Acheive[DriverForwardAccelOverrideAccelRequest] **InformalDef:** *If a) a) the vehicle is moving in the forward direction and* *b) the driver is applying the brake pedal or the throttle pedal* *then the subsystem a) shall not request acceleration or* *b) shall request acceleration less than -2 m/s^2 (i.e., request a "hard" stop of the vehicle) .* **FormalDef:** $\forall$ ar: AccelerationRequest, vsp: VehicleSpeed, sn: SubsystemName, hmi: HumanMachineInterface, tp:ThrottlePedal, st: StoppedTime, gt: GoTime $\bullet$ (InForwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active)) $\Rightarrow$ ($\neg$ ar.active $\vee$ ((ar.value < -2 m/s^2) $\wedge$ (ar.value < RequestedAcceleration(bp, tp))))

Figure C.21. ICPA for Achieve[DriverForwardAccelOverride] (4 of 4)

ICPA for Achieve[DriverBackwardAccelOverride] (1 of 4)

System Safety Goal

Goal: Achieve[DriverBackwardAccelOverride]

InformalDef: *If a) the vehicle is moving in the backward direction and*
b) the driver is applying the brake pedal or the throttle pedal and
c) a subsystem is requesting a vehicle acceleration less than or equal to 2 m/s^2 (i.e., not requesting a "hard" stop of the vehicle),
then the subsystem shall not control vehicle acceleration.

FormalDef: $\forall$ va: VehicleAcceleration, sn: SubsystemName, bp: BrakePedal, tp: ThrottlePedal
$\bullet\!\!\bullet$ (InBackwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active) $\wedge$ RequestingAcceleration(sn)
$\wedge$ (RequestedAcceleration(sn) $\leq$ 2 m/s^2)) $\Rightarrow \neg$ (va.source = sn)

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	Arbiter	ac: AccelerationCommand	❑ (va.value = $\bullet$ ac.value) % Acceleration is equal to the previous acceleration command value	01
			❑ (va.source = $\bullet$ ac.source) % The source of acceleration is equal to the source of the previous acceleration command	02
			❑ (ac.source $\in$ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	03

Figure C.22. ICPA for Achieve[DriverBackwardAccelOverride] (1 of 4)

ICPA for Achieve[DriverBackwardAccelOverride] (2 of 4)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	04
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	05
			● ¬ ar.active ⇒ ¬ (ac.source = ● ar.source) % If a previous acceleration request is not active, then the source of % an acceleration command is not the source of that previous % acceleration request	06
sn	Arbiter		❑ (sn ∈ {Arbiter, CA, RCA, ACC, LCA, PA}) % A subsystem is either the arbiter or one of the feature subsystems	07
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	((ar.source = sn) ∧ ar.active) ⇔ RequestingAcceleration(sn) % If an acceleration request is active, then the source of that acceleration request is requesting acceleration, and vice versa	08
			((ar.source = sn) ∧ ar.active) ⇔ (RequestedAcceleration(sn) = ar.value) % If an acceleration request is active, then the requested acceleration % of the source of the acceleration request is the value of the % acceleration request.	09

Figure C.23. ICPA for Achieve[DriverBackwardAccelOverride] (2 of 4)

ICPA for Achieve[DriverBackwardAccelOverride] (3 of 4)	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter, Secondary: CA, RCA, ACC, LCA, PA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Also, OR reduction is used for CA, RCA, ACC, LCA, and PA goals, and restricts acceleration requests one state earlier than necessary.)
Goal Elaboration	
●● (InBackwardMotion(vsp.value) ∧ (bp.active ∨ tp.active) ∧ RequestingAcceleration(sn) ∧ (RequestedAcceleration(sn) ≤ 2 m/s^2)) ⇒ ¬ (va.source = sn)	Indirect Control Relationships
● (InBackwardMotion(vsp.value) ∧ (bp.active ∨ tp.active) ∧ ((ar.source = sn) ∧ ar.active) ∧ (ar.value ≤ 2 m/s^2)) ⇒ ¬ (ac.source = ● ar.source)	01, 02, 08, 09 – Introduce accuracy/actuation goal tactic. Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02)
● (InBackwardMotion(vsp.value) ∧ (bp.active ∨ tp.active)) ⇒ (¬ ar.active ∨ (ar.value > 2 m/s^2))	OR reduction. Also restricts acceleration requests one state earlier than necessary.

Figure C.24. ICPA for Achieve[DriverBackwardAccelOverride] (3 of 4)

ICPA for Achieve[DriverBackwardAccelOverride] (4 of 4)

Subsystem Safety Goals

Subsystem: Arbiter
Controls: AcelerationCommand
Observes: AccelerationRequest, BrakePedal, ThrottlePedal
Goal: Acheive[DriverBackwardAccelOverrideAccelCommand]
InformalDef: *If a) the vehicle is moving in the backward direction and*
b) the driver is applying the brake pedal or the throttle pedal and
c) a subsystem is requesting a vehicle acceleration less than or equal to 2 m/s^2 (i.e., not requesting a "hard" stop of the vehicle),
then the source of the acceleration command shall not be a subsystem
FormalDef: $\forall$ ac: AccelerationCommand, bp: BrakePedal, ar:AccelerationRequest, tp:ThrottlePedal
$\bullet$ (InBackwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active) $\wedge$ ((ar.source = sn) $\wedge$ ar.active) $\wedge$ (ar.value $\leq$ 2 m/s^2))
$\Rightarrow \neg$ (ac.source = $\bullet$ ar.source)

Subsystem: CA, RCA, ACC, LCA, PA
Controls: AcelerationRequest
Observes: BrakePedal, ThrottlePedal
Goal: Acheive[DriverBackwardAccelOverrideAccelRequest]
InformalDef: *If a) a) the vehicle is moving in the backward direction and*
b) the driver is applying the brake pedal or the throttle pedal
then the subsystem a) shall not request acceleration or
b) shall request acceleration greater than 2 m/s^2 (i.e., request a "hard" stop of the vehicle) .
FormalDef: $\forall$ ar: AccelerationRequest, vsp: VehicleSpeed, sn: SubsystemName, hmi: HumanMachineInterface, tp:ThrottlePedal, st: StoppedTime, gt: GoTime
$\bullet$ (InBackwardMotion(vsp.value) $\wedge$ (bp.active $\vee$ tp.active))
$\Rightarrow$ ($\neg$ ar.active $\vee$ ((ar.value > 2 m/s^2) $\wedge$ (ar.value < RequestedAcceleration(bp, tp))))

Figure C.25. ICPA for Achieve[DriverBackwardAccelOverride] (4 of 4)

ICPA for Achieve[DriverSteeringOverride] (1 of 4)

System Safety Goal

Goal: Achieve[DriverSteeringOverride]
InformalDef: *If the driver is turning the steering wheel, then no subsystem shall control vehicle steering*
FormalDef: ∀ vst: VehicleSteering, sn: SubsystemName, sw: SteeringWheel
●● (sw.active) ⇒ ¬ (vst.source = sn)

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
vst	Arbiter	sc: SteeringCommand	❑ (vst.value = ● sc.value) % The value of steering is equal to the value of the previous steering % command	01
			❑ (vst.source = ● sc.source) % The source of steering is the source of the previous steering % command	02
			❑ (sc.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of a steering command is either the driver or one of the % feature subsystems	03

Figure C.26. ICPA for Achieve[DriverSteeringOverride] (1 of 4)

ICPA for Achieve[DriverSteeringOverride] (2 of 4)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
vst	CA, RCA, ACC, LCA, PA	sr: SteeringRequest	sc.source = ● sr.source ⇒ sc.value = ● sr.value % If the source of a steering command is the source of a previous % steering request, then the value of the steering command is equal % to the value of the previous steering request	04
			❑ (sr.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of a steering request is one of the feature subsystems	05
			● ¬ sr.active ⇒ ¬ (sc.source = ● sr.source) % If a previous steering request is not active, then the source of a % steering command is not the source of that previous steering % request	06
sn	Arbiter		❑ (sn ∈ {Arbiter, CA, RCA, ACC, LCA, PA}) % A subsystem is either the arbiter or one of the feature subsystems	07
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	((sr.source = sn) ∧ sr.active) ⇔ RequestingSteering(sn) % If a steering request is active, then the source of that steering request is requesting steering, and vice versa	08
			((sr.source = sn) ∧ sr.active) ⇔ (RequestedSteering(sn) = sr.value) % If a steering request is active, then the steering value requested by % the source of the steering request is the value of the steering % request	09

Figure C.27. ICPA for Achieve[DriverSteeringOverride] (2 of 4)

ICPA for Achieve[DriverSteeringOverride] (3 of 4)	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter, Secondary: CA, RCA, ACC, LCA, PA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Restricts acceleration requests one state earlier than necessary.)
Goal Elaboration	
●● (sw.active) ⇒ ¬ (vst.source = sn)	Indirect Control Relationships
● (sw.active) ⇒ ¬ (sc.source = ● sr.source)	01, 02, 03, 05 – Introduce accuracy/actuation goal tactic. Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02)
● (sw.active) ⇒ ¬ sr.active	06 – Split Lack of Monitorability/Controllability by Chaining. Restricts acceleration requests one state earlier than necessary.

Figure C.28. ICPA for Achieve[DriverSteeringOverride] (3 of 4)

ICPA for Achieve[DriverSteeringOverride] (4 of 4)
Subsystem Safety Goals
Subsystem: Arbiter **Controls:** SteeringCommand **Observes:** SteeringRequest, SteeringWheel **Goal:** Acheive[DriverSteeringOverrideSteeringCommand] **InformalDef:** *If the driver is turning the steering wheel, then the source of steering commands shall not be a steering request from a subsytem.* **FormalDef:** $\forall$ sc: SteeringCommand, sr: SteeringRequest, sw: SteeringWheel $\bullet$ (sw.active) $\Rightarrow \neg$ (sc.source = $\bullet$ sr.source)
Subsystem: CA, RCA, ACC, LCA, PA **Controls:** SteeringRequest **Observes:** SteeringWheel **Goal:** Acheive[DriverSteeringOverrideSteeringRequest] **InformalDef:** *If the driver is turning the steering wheel, then subsystem shall not request steering* **FormalDef:** $\forall$ sr: SteeringRequest, sw: SteeringWheel $\bullet$ (sw.active) $\Rightarrow \neg$ sr.active

Figure C.29. ICPA for Achieve[DriverSteeringOverride] (4 of 4)

ICPA for Achieve[ForwardBlockAccelSteering] (1 of 4)

System Safety Goal

Goal: Achieve[ForwardBlockAccelSteering]
InformalDef: *If the vehicle is moving forward, then the subsystem RCA shall not control vehicle acceleration or steering*
FormalDef: ∀ va: VehicleAcceleration, vst: VehicleSteering, vsp: VehicleSpeed
●● InForwardMotion(vsp.value) ⇒ ¬ ((va.source = 'RCA') ∨ (vst.source = 'RCA'))

Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	Arbiter	ac: AccelerationCommand	❑ (va.value = ● ac.value) % Acceleration is equal to the previous acceleration command value	01
			❑ (va.source = ● ac.source) % The source of acceleration is equal to the source of the previous acceleration command	02
			❑ (ac.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	03
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	04
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	05
			● ¬ ar.active ⇒ ¬ (ac.source = ● ar.source) % If a previous acceleration request is not active, then the source of % an acceleration command is not the source of that previous % acceleration request	06

Figure C.30. ICPA for Achieve[ForwardBlockAccelSteering] (1 of 4)

ICPA for Achieve[ForwardBlockAccelSteering] (2 of 4)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
vst	Arbiter	sc: SteeringCommand	❑ (vst.value = ● sc.value) % The value of steering is equal to the value of the previous steering % command	07
			❑ (vst.source = ● sc.source) % The source of steering is the source of the previous steering % command	08
			❑ (sc.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of a steering command is either the driver or one of the % feature subsystems	09
	CA, RCA, ACC, LCA, PA	sr: SteeringRequest	sc.source = ● sr.source ⇒ sc.value = ● sr.value % If the source of a steering command is the source of a previous % steering request, then the value of the steering command is equal % to the value of the previous steering request	10
			❑ (sr.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of a steering request is one of the feature subsystems	11
			● ¬ sr.active ⇒ ¬ (sc.source = ● sr.source) % If a previous steering request is not active, then the source of a % steering command is not the source of that previous steering % request	12

Figure C.31. ICPA for Achieve[ForwardBlockAccelSteering] (2 of 4)

ICPA for Achieve[ForwardBlockAccelSteering] (3 of 4)	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter, Secondary: RCA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Restricts acceleration requests one state earlier than necessary)
Goal Elaboration	
●● InForwardMotion(vsp.value) ⇒ ¬ ((va.source = 'RCA') ∨ (vst.source = 'RCA'))	Indirect Control Relationships
● InForwardMotion(vsp.value) ⇒ ¬ ((ac.source = 'RCA') ∨ (sc.source = 'RCA'))	02, 08 – Introduce accuracy/actuation goal tactic. Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02)
● InForwardMotion(vsp.value) ⇒ ¬ (ar.active ∨ sr.active)	06, 12 – Split Lack of Monitorability/Controllability by Chaining. Also restricts acceleration requests one state earlier than necessary

Figure C.32. ICPA for Achieve[ForwardBlockAccelSteering] (3 of 4)

ICPA for Achieve[ForwardBlockAccelSteering] (4 of 4)
Subsystem Safety Goals
Subsystem: Arbiter **Controls:** AccelerationCommand, SteeringCommand **Observes:** VehicleSpeed **Goal:** Acheive[ForwardBlockAccelSteeringCommand] **InformalDef:** *If the vehicle is moving forward, then the source of acceleration commands and steering commands shall not be RCA* **FormalDef:** $\forall$ ac: AccelerationCommand, sc: SteeringCommand, vsp: VehicleSpeed ● InForwardMotion(vsp.value) $\Rightarrow \neg$ ((ac.source = 'RCA') $\vee$ (sc.source = 'RCA'))
Subsystem: RCA **Controls:** AccelerationRequest, SteeringRequest **Observes:** VehicleSpeed **Goal:** Acheive[ForwardBlockAccelSteeringRequest] **InformalDef:** *If the vehicle is moving forward, then RCA a) shall not request acceleration and b) shall not request steering* **FormalDef:** $\forall$ ar: AccelerationRequest, sr: SteeringRequest, vsp: VehicleSpeed ● InForwardMotion(vsp.value) $\Rightarrow \neg$ (ar.active $\vee$ sr.active)

Figure C.33. ICPA for Achieve[ForwardBlockAccelSteering] (4 of 4)

ICPA for Achieve[BackwardBlockAccelSteering] (1 of 5)
System Safety Goal
Goal: Achieve[BackwardBlockAccelSteering] **InformalDef:** *If the vehicle is moving backward, then the subsystems CA, ACC, and LCA shall not control vehicle acceleration or steering.* **FormalDef:** $\forall$ va: VehicleAcceleration, vst: VehicleSteering, vsp: VehicleSpeed $\bullet\bullet$ InBackwardMotion(vsp.value) $\Rightarrow \neg$ ((va.source $\in$ {CA, ACC, LCA}) $\vee$ (vst.source $\in$ {CA, ACC, LCA}))

Figure C.34. ICPA for Achieve[BackwardBlockAccelSteering] (1 of 5)

ICPA for Achieve[BackwardBlockAccelSteering] (2 of 5)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
va	Arbiter	ac: AccelerationCommand	❑ (va.value = ● ac.value) % Acceleration is equal to the previous acceleration command value	01
			❑ (va.source = ● ac.source) % The source of acceleration is equal to the source of the previous acceleration command	02
			❑ (ac.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of an acceleration command is either the driver or one % of the feature subsystems	03
	CA, RCA, ACC, LCA, PA	ar: AccelerationRequest	ac.source = ● ar.source ⇒ ac.value = ● ar.value % If the source of an acceleration command is the source of a % previous acceleration request, then the value of the acceleration % command is equal to the value of the previous acceleration request	04
			❑ (ar.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of an acceleration request is one of the feature % subsystems	05
			● ¬ ar.active ⇒ ¬ (ac.source = ● ar.source) % If a previous acceleration request is not active, then the source of % an acceleration command is not the source of that previous % acceleration request	06

Figure C.35. ICPA for Achieve[BackwardBlockAccelSteering] (2 of 5)

ICPA for Achieve[BackwardBlockAccelSteering] (3 of 5)				
Variable	Indirect Control Path		Indirect Control Relationships	#
	Subsystem	Variables		
vst	Arbiter	sc: SteeringCommand	❑ (vst.value = ● sc.value) % The value of steering is equal to the value of the previous steering % command	07
			❑ (vst.source = ● sc.source) % The source of steering is the source of the previous steering % command	08
			❑ (sc.source ∈ {Driver, CA, RCA, ACC, LCA, PA}) % The source of a steering command is either the driver or one of the % feature subsystems	09
	CA, RCA, ACC, LCA, PA	sr: SteeringRequest	sc.source = ● sr.source ⇒ sc.value = ● sr.value % If the source of a steering command is the source of a previous % steering request, then the value of the steering command is equal % to the value of the previous steering request	10
			❑ (sr.source ∈ {CA, RCA, ACC, LCA, PA}) % The source of a steering request is one of the feature subsystems	11
			● ¬ sr.active ⇒ ¬ (sc.source = ● sr.source) % If a previous steering request is not active, then the source of a % steering command is not the source of that previous steering % request	12

Figure C.36. ICPA for Achieve[BackwardBlockAccelSteering] (3 of 5)

ICPA for Achieve[BackwardBlockAccelSteering] (4 of 5)	
Goal Coverage Strategy	
Goal Assignment	Redundant Responsibility (Primary: Arbiter, Secondary: CA, ACC, LCA)
Goal Scope	Restrictive (Assumes worst-case vehicle response to acceleration command; real response may be different. Restricts acceleration requests one state earlier than necessary.)
Goal Elaboration	
●● InBackwardMotion(vsp.value) ⇒ ¬ ((va.source ∈ {CA, ACC, LCA}) ∨ (vst.source ∈ {CA, ACC, LCA}))	Indirect Control Relationships
● InBackwardMotion(vsp.value) ⇒ ¬ ((ac.source ∈ {CA, ACC, LCA}) ∨ (sc.source ∈ {CA, ACC, LCA}))	02, 08 – Introduce accuracy/actuation goal tactic. Assumes worst-case actuation delays for source of acceleration and acceleration value (built-in to definition of 01, 02)
● InBackwardMotion(vsp.value) ⇒ ¬ (ar.active ∨ sr.active)	06, 12 – Split Lack of Monitorability/Controllability by Chaining. Also, restricts acceleration requests one state earlier than necessary.

Figure C.37. ICPA for Achieve[BackwardBlockAccelSteering] (4 of 5)

ICPA for Achieve[BackwardBlockAccelSteering] (5 of 5)
Subsystem Safety Goals
Subsystem: Arbiter **Controls:** AccelerationCommand, SteeringCommand **Observes:** VehicleSpeed **Goal:** Acheive[ForwardBlockAccelSteeringCommand] **InformalDef:** *If the vehicle is moving backward, then the source of acceleration commands and steering commands shall not be* CA, ACC, or LCA. **FormalDef:** $\forall$ ac:AccelerationCommand, sc: SteeringCommand, vsp: VehicleSpeed ● InBackwardMotion(vsp.value) $\Rightarrow \neg$ ((ac.source $\in$ {CA, ACC, LCA}) $\vee$ (sc.source $\in$ {CA, ACC, LCA}))
Subsystem: RCA **Controls:** AccelerationRequest, SteeringRequest **Observes:** VehicleSpeed **Goal:** Acheive[BackwardBlockAccelSteeringRequest] **InformalDef:** *If the vehicle is moving forward, then CA, ACC, and LCA a) shall not request acceleration and b) shall not request steering* **FormalDef:** $\forall$ ar:AccelerationRequest, sr: SteeringRequest, vsp: VehicleSpeed ● InBackwardMotion(vsp.value) $\Rightarrow \neg$ (ar.active $\vee$ sr.active)

Figure C.38. ICPA for Achieve[BackwardBlockAccelSteering] (5 of 5)

Appendix D

Evaluation Scenario Results

Table D.1. Goal and subgoal violations for Scenario 1

Scenario 1: The host vehicle is travelling forward, starting from a stop 20 m behind another stopped vehicle. ACC is enabled, but not engaged. CA is enabled.						
Notes: The simulation terminated early at time 12.681 s.						
		First	Last	Longest	Shortest	Total #
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	12.589	12.589	12.589	12.589	1
	Duration (sec)	0.004	0.004	0.004	0.004	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	12.583	12.68	12.583	12.68	6
	Duration (sec)	0.008	0.001	0.008	0.001	
CA	**Start (sec)**	12.6	12.6	12.6	12.6	1
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	9.624	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
None						
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.119	0.119	0.119	0.119	
Goal 5: Achieve[DriverForwardAccelOverride]						
None						
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
None						

Table D.2. Goal and subgoal violations for Scenario 2

Scenario 2: The host vehicle is travelling forward, starting from a stop 20 m behind another stopped vehicle. ACC is enabled, but not engaged. CA is enabled. Just after CA begins to perform an emergency braking action at time 12.55 s to avoid the stopped vehicle, the driver engages PA at time 12.56 s.						
Notes: Simulation terminated early at time 12.588 s.						
		First	**Last**	**Longest**	**Shortest**	**Total #**
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	12.587	12.587	12.587	12.587	1
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	12.568	12.568	12.568	12.568	1
	Duration (sec)	0.02	0.02	0.02	0.02	
Arbiter	**Start (sec)**	12.561	12.561	12.561	12.561	1
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	9.624	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
Vehicle	**Start (sec)**	12.561	12.561	12.561	12.561	1
	Duration (sec)	0.027	0.027	0.027	0.027	
Arbiter	**Start (sec)**	12.561	12.561	12.561	12.561	1
	Duration (sec)	0.027	0.027	0.027	0.027	
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.119	0.119	0.119	0.119	
Goal 5: Achieve[DriverForwardAccelOverride]						
None						
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
None						

Table D.3. Goal and subgoal violations for Scenario 3

Scenario 3: The host vehicle is travelling forward, starting from a stop 20 m behind another stopped vehicle. ACC is enabled, but not engaged. CA is enabled. Just after CA begins to perform an emergency braking action at time 12.55 s to avoid the stopped vehicle, the driver applies the throttle pedal at time 12.56 s.						
Notes: Simulation terminated normally at time 20 s.						
		First	**Last**	**Longest**	**Shortest**	**Total #**
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	13.652	13.652	13.652	13.652	1
	Duration (sec)	0.003	0.003	0.003	0.003	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	12.565	12.913	12.913	12.565	3
	Duration (sec)	0.002	0.004	0.004	0.002	
CA	**Start (sec)**	12.6	13.85	12.6	12.6	4
	Duration (sec)	0.001	0.001	0.001	0.001	
ACC	**Start (sec)**	12.75	15.6	12.75	12.75	47
	Duration (sec)	0.001	0.001	0.001	0.001	
LCA	**Start (sec)**	12.75	15.6	12.75	12.75	47
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	9.624	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
None						
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.119	0.119	0.119	0.119	
Goal 5: Achieve[DriverForwardAccelOverride]						
Vehicle	**Start (sec)**	12.562	13.652	12.902	12.562	4
	Duration (sec)	0.04	0.063	0.061	0.04	
CA	**Start (sec)**	12.6	13.85	13.85	12.6	5
	Duration (sec)	0.05	0.2	0.2	0.05	
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
None						

Table D.4. Goal and subgoal violations for Scenario 4

Scenario 4: The host vehicle is travelling forward, starting 25m behind another vehicle that is travelling between 10-25 k/h (2.78-6.94 m/s). The driver accelerates the host vehicle by applying the throttle pedal from time 0.5 s to 8.5 s. At time 2.0 s, ACC is engaged by the driver.						
Notes: Simulation terminated early at time 13.142 s.						
		First	**Last**	**Longest**	**Shortest**	**Total #**
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	12.987	13.127	13.002	13.074	10
	Duration (sec)	0.004	0.014	0.067	0.001	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	8.617	13.127	12.997	11.248	98
	Duration (sec)	0.004	0.006	0.039	0.001	
Arbiter	**Start (sec)**	9.051	13.051	9.051	9.051	48
	Duration (sec)	0.001	0.001	0.001	0.001	
ACC	**Start (sec)**	2.05	13.05	2.05	2.05	49
	Duration (sec)	0.001	0.001	0.001	0.001	
LCA	**Start (sec)**	2.05	13.05	2.05	2.05	49
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	3.906	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
None						
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.167	0.167	0.167	0.167	
Goal 5: Achieve[DriverForwardAccelOverride]						
Vehicle	**Start (sec)**	2.052	2.052	2.052	2.052	1
	Duration (sec)	0.05	0.05	0.05	0.05	
Arbiter	**Start (sec)**	2.051	2.051	2.051	2.051	1
	Duration (sec)	0.05	0.05	0.05	0.05	
ACC	**Start (sec)**	2	2	2	2	1
	Duration (sec)	0.05	0.05	0.05	0.05	
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
Vehicle	**Start (sec)**	13.074	13.074	13.074	13.074	1
	Duration (sec)	0.002	0.002	0.002	0.002	
Arbiter	**Start (sec)**	13.074	13.074	13.074	13.074	1
	Duration (sec)	0.002	0.002	0.002	0.002	
ACC	**Start (sec)**	13.074	13.074	13.074	13.074	1
	Duration (sec)	0.002	0.002	0.002	0.002	

Table D.5. Goal and subgoal violations for Scenario 5

Scenario 5: The host vehicle is travelling forward, starting 25m behind another vehicle that is travelling between 10-25 k/h (2.78-6.94 m/s). The driver accelerates the host vehicle by applying the throttle pedal from time 0.5 s to 2.5 s. At time 2.0 s, ACC is engaged by the driver. The driver applies the brake pedal from time 4.0 s to time 9.0 s.						
Notes: Simulation terminated normally at time 20 s.						
		First	Last	Longest	Shortest	Total #
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	2.641	2.731	2.641	2.653	6
	Duration (sec)	0.009	0.005	0.009	0.005	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	2.638	3.41	2.726	2.837	28
	Duration (sec)	0.006	0.001	0.008	0.001	
Arbiter	**Start (sec)**	2.651	4.001	2.651	2.651	28
	Duration (sec)	0.001	0.001	0.001	0.001	
ACC	**Start (sec)**	2.05	4	2.05	2.05	30
	Duration (sec)	0.001	0.001	0.001	0.001	
LCA	**Start (sec)**	2.05	4	2.05	2.05	30
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	4.123	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
None						
Goal 4: Achieve[NoAutoAccelFromStop]						
ACC	**Start (sec)**	5.916	6.302	5.916	6.302	2
	Duration (sec)	0.103	0.048	0.103	0.048	
LCA	**Start (sec)**	5.916	6.302	5.916	6.302	2
	Duration (sec)	0.103	0.048	0.103	0.048	
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.167	0.167	0.167	0.167	
Goal 5: Achieve[DriverForwardAccelOverride]						
Vehicle	**Start (sec)**	2.052	4.002	2.052	2.052	2
	Duration (sec)	0.05	0.05	0.05	0.05	
Arbiter	**Start (sec)**	2.051	4.001	2.051	2.051	2
	Duration (sec)	0.05	0.05	0.05	0.05	
ACC	**Start (sec)**	2	2	2	2	1
	Duration (sec)	0.05	0.05	0.05	0.05	
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
None						

Table D.6. Goal and subgoal violations for Scenario 6 (1 of 2)

Scenario 6: The host vehicle is travelling forward, starting 25m behind another vehicle that is travelling between 10-25 k/h (2.78-6.94 m/s). The driver accelerates the host vehicle by applying the throttle pedal from time 0.5 s to 2.5 s. At time 2.0 s, ACC is engaged by the driver. LCA is enabled by the driver at time 4.0 s, and engaged at time 6.0 s. (1 of 2)						
Notes: Simulation terminated early at time 13.954 s.						
		First	Last	Longest	Shortest	Total #
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	Start (sec)	2.641	13.953	2.641	13.911	17
	Duration (sec)	0.009	0.001	0.009	0.001	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	Start (sec)	2.638	13.953	13.828	2.837	108
	Duration (sec)	0.006	0.001	0.038	0.001	
Arbiter	Start (sec)	2.651	13.951	2.651	2.651	83
	Duration (sec)	0.001	0.001	0.001	0.001	
ACC	Start (sec)	2.05	13.95	2.05	2.05	85
	Duration (sec)	0.001	0.001	0.001	0.001	
LCA	Start (sec)	2.05	13.95	2.05	2.05	85
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	Start (sec)	0.001	4.12	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
Vehicle	Start (sec)	5.052	5.052	5.052	5.052	1
	Duration (sec)	8.902	8.902	8.902	8.902	
Arbiter	Start (sec)	5.052	5.052	5.052	5.052	1
	Duration (sec)	8.902	8.902	8.902	8.902	
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	Start (sec)	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.167	0.167	0.167	0.167	
Goal 5: Achieve[DriverForwardAccelOverride]						
Vehicle	Start (sec)	2.052	2.052	2.052	2.052	1
	Duration (sec)	0.05	0.05	0.05	0.05	
Arbiter	Start (sec)	2.051	2.051	2.051	2.051	1
	Duration (sec)	0.05	0.05	0.05	0.05	
ACC	Start (sec)	2	2	2	2	1
	Duration (sec)	0.05	0.05	0.05	0.05	

Table D.7. Goal and subgoal violations for Scenario 6 (2 of 2)

Scenario 6: The host vehicle is travelling forward, starting 25m behind another vehicle that is travelling between 10-25 k/h (2.78-6.94 m/s). The driver accelerates the host vehicle by applying the throttle pedal from time 0.5 s to 2.5 s. At time 2.0 s, ACC is engaged by the driver. LCA is enabled by the driver at time 4.0 s, and engaged at time 6.0 s. (2 of 2)						
Notes: Simulation terminated early at time 13.954 s.						
		First	**Last**	**Longest**	**Shortest**	**Total #**
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
Vehicle	**Start (sec)**	13.905	13.905	13.905	13.905	1
	Duration (sec)	0.049	0.049	0.049	0.049	
Arbiter	**Start (sec)**	13.905	13.905	13.905	13.905	1
	Duration (sec)	0.049	0.049	0.049	0.049	
ACC	**Start (sec)**	13.905	13.905	13.905	13.905	1
	Duration (sec)	0.049	0.049	0.049	0.049	
LCA	**Start (sec)**	13.905	13.905	13.905	13.905	1
	Duration (sec)	0.049	0.049	0.049	0.049	

Table D.8. Goal and subgoal violations for Scenario 7

Scenario 7: The host vehicle is travelling in reverse, 30 m in front of another stopped vehicle, with RCA enabled.						
Notes: Simulation terminated normally at 20 s.						
		First	**Last**	**Longest**	**Shortest**	**Total #**
Goal 1: Maintain[AutoAccelBelowThreshold]						
None						
Goal 2: Maintain[AutoJerkBelowThreshold]						
ACC	**Start (sec)**	8.45	15.6	8.45	8.45	42
	Duration (sec)	0.001	0.001	0.001	0.001	
LCA	**Start (sec)**	8.45	15.6	8.45	8.45	42
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	6.264	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
None						
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.044	0.044	0.044	0.044	
Goal 5: Achieve[DriverForwardAccelOverride]						
None						
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
None						

Table D.9. Goal and subgoal violations for Scenario 8

Scenario 8: The host vehicle is travelling in reverse, 30 m in front of another stopped vehicle. ACC is enabled and engaged by the driver at time 2.0 s.						
Notes: Simulation terminated early at time 2.083 s.						
		First	Last	Longest	Shortest	Total #
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	2.08	2.08	2.08	2.08	1
	Duration (sec)	0.003	0.003	0.003	0.003	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	2.055	2.083	2.07	2.083	3
	Duration (sec)	0.008	0.001	0.011	0.001	
ACC	**Start (sec)**	2.05	2.05	2.05	2.05	1
	Duration (sec)	0.001	0.001	0.001	0.001	
LCA	**Start (sec)**	2.05	2.05	2.05	2.05	1
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	0.001	0.001	0.001	1
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
None						
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	0.044	0.044	0.044	0.044	
Goal 5: Achieve[DriverForwardAccelOverride]						
None						
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
Vehicle	**Start (sec)**	2.051	2.051	2.051	2.051	1
	Duration (sec)	0.032	0.032	0.032	0.032	
Arbiter	**Start (sec)**	2.051	2.051	2.051	2.051	1
	Duration (sec)	0.032	0.032	0.032	0.032	
ACC	**Start (sec)**	2	2	2	2	1
	Duration (sec)	0.083	0.083	0.083	0.083	

Table D.10. Goal and subgoal violations for Scenario 9

Scenario 9: The host vehicle is stopped 20 m behind another stopped vehicle. PA is enabled and engaged by the driver at time 2.0 s.						
Notes: Simulation terminated early at time 2.113 s.						
		First	**Last**	**Longest**	**Shortest**	**Total #**
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	2.098	2.098	2.098	2.098	1
	Duration (sec)	0.015	0.015	0.015	0.015	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	2.003	2.112	2.042	2.003	6
	Duration (sec)	0.001	0.001	0.046	0.001	
PA	**Start (sec)**	0.001	0.001	0.001	0.001	1
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
Vehicle	**Start (sec)**	2.001	2.001	2.001	2.001	1
	Duration (sec)	0.112	0.112	0.112	0.112	
Arbiter	**Start (sec)**	2.001	2.001	2.001	2.001	
	Duration (sec)	0.112	0.112	0.112	0.112	
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	2.093	2.093	2.093	2.093	
Goal 5: Achieve[DriverForwardAccelOverride]						
None						
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
None						

Table D.11. Goal and subgoal violations for Scenario 10

Scenario 10: The host vehicle is stopped 20 m behind another stopped vehicle, ACC is enabled and engaged by the driver at time 2.0 s.						
Notes: Simulation terminated early at time 14.625 s.						
		First	**Last**	**Longest**	**Shortest**	**Total #**
Goal 1: Maintain[AutoAccelBelowThreshold]						
Vehicle	**Start (sec)**	14.589	14.589	14.589	14.589	1
	Duration (sec)	0.004	0.004	0.004	0.004	
Goal 2: Maintain[AutoJerkBelowThreshold]						
Vehicle	**Start (sec)**	14.583	14.598	14.583	14.598	2
	Duration (sec)	0.008	0.002	0.008	0.002	
CA	**Start (sec)**	14.6	14.6	14.6	14.6	1
	Duration (sec)	0.001	0.001	0.001	0.001	
PA	**Start (sec)**	0.001	11.538	0.001	0.001	2
	Duration (sec)	0.001	0.001	0.001	0.001	
Goal 3: Achieve[SubsystemAccelSteeringAgreement]						
Goal 4: Achieve[NoAutoAccelFromStop]						
PA	**Start (sec)**	0.01	0.01	0.01	0.01	1
	Duration (sec)	2.133	2.133	2.133	2.133	
Goal 5: Achieve[DriverForwardAccelOverride]						
None						
Goal 6: Achieve[DriverBackwardAccelOverride]						
None						
Goal 7: Achieve[DriverSteeringOverride]						
None						
Goal 8: Achieve[ForwardBlockAccelSteering]						
None						
Goal 9: Achieve[BackwardBlockAccelSteering]						
None						

Bibliography

[1] (2009) The Carsim® website. Accessed on April 6, 2009. [Online]. Available: http://www.carsim.com/

[2] (2009) The MathWorks™ Simulink® website. Accessed on April 6, 2009. [Online]. Available: http://www.mathworks.com/products/simulink/

[3] (2009) The National Instruments LabVIEW website. Accessed on April 6, 2009. [Online]. Available: http://www.ni.com/labview/

[4] E. A. Addy, "A case study on isolation of safety-critical software," in *Proceedings of the 6th Annual Conference on Computer Assurance (COMPASS'91)*, June 1991, pp. 75–83.

[5] Aristotle, *Metaphysics: Book VIII*. Trans. by W.D. Ross, 350 B.C.E., accessed on April 6, 2009. [Online]. Available: http://classics.mit.edu/Aristotle/metaphysics.8.viii.html

[6] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1972.

[8] P. Bishop and R. Bloomfield, "A methodology for safety case development," in *Proceedings of the 6th Safety-Critical Systems Symposium (SSS'98)*, Birmingham, UK, Feb. 1998.

[9] F. Bitsch, "Classification of safety requirements for formal verification of software models for industrial automation systems," in *Proceedings of the International Conference on Software and Systems Engineering and their Applications (ICSSEA 2000)*, Paris, France, Dec. 2000.

[10] F. Bitsch, "Safety patterns - the key to formal specification of safety requirements," in *Proceedings of the 20th Intl. Conference on Computer Safety, Reliability and Security (SAFECOMP '01)*, ser. Lecture Notes in Computer Science, U. Voges, Ed., vol. 2187. Budapest, Hungary: Springer-Verlag, Sept. 2001, pp. 176–189.

[11] J. Black and P. Koopman, "Indirect control path analysis and goal coverage strategies for elaborating system safety goals in composite systems," in *Proceedings of the 14th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'08)*, Taipei, Taiwan, Dec. 2008, pp. 184–191.

[12] J. Black and P. Koopman, "System safety as an emergent property in composite systems," in *Proceedings of the* 39th *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'09)*, Estoril, Portugal, June 2009, p. (in press).

[13] T. F. Bowen, F. S. Dworack, C. H. Chow, N. Griffeth, G. E. Herman, and Y.-J. Lin, "The feature interaction problem in telecommunications systems," in *Proceedings of the 7th International Conference on Software Engineering for Telecom. Switching Sys.*, Bournemouth, UK, July 1989, pp. 59–62.

[14] M. Brown and N. G. Leveson, "Modeling controller tasks for safety analysis," in *Proceedings of the* 2nd *Workshop on Human Error, Safety, and System Development (HESSD'98)*, Seattle, WA, USA, Apr. 1998, pp. 80–89.

[15] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen, "A feature-interaction benchmark for IN and beyond," *IEEE Communications Magazine*, vol. 31, no. 3, pp. 64–69, Mar. 1993.

[16] P. Cariani, "Emergence and artificial life," in *Artificial Life II: Proceedings of the 2nd Workshop on Artificial Life*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds. Addison-Wesley, 1991, pp. 775–797.

[17] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[18] A. Dardenne, A. v. Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," in *Science of Computer Programming*, M. Sintzoff, C. Ghezzi, and G. Roman, Eds. Amsterdam, The Netherlands: Elsevier Science, Apr. 1993, no. 1-2, pp. 3–50.

[19] R. Darimont, "Process support for requirements elaboration," Ph.D. dissertation, University catholique de Louvain, Louvain, Belgium, June 1995, accessed on April 6, 2009. [Online]. Available: http://www.info.ucl.ac.be/Research/Publication/Theses/darimont-phd.pdf

[20] R. Darimont and A. v. Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," in *Proceedings of the* 4th *ACM SIGSOFT Symposium on Foundations of Software Engineering*, ser. Software Engineering Notes, vol. 21, no. 6. ACM SIGSOFT, Nov. 1996, pp. 179–190.

[21] V. Darley, "Emergent phenomena and complexity," in *Artificial Life IV: Proceedings of the 4th Workshop on Synthesis and Simulation of Living Systems*, R. A. Brooks and P. Maes, Eds. MIT Press, 1994, pp. 411–416.

[22] J. Dehlinger and R. R. Lutz, "Software fault tree analysis for product lines," in *Proceedings of the* 8th *IEEE International Symposium on High Assurance Systems Engineering (HASE04)*, Tampa, FL, USA, Mar. 2004, pp. 12–21.

[23] *DOE Guideline: Root Cause Analysis Guidance Document*, Department of Energy (DOE) Office of Nuclear Energy and Office of Nuclear Safety Policy and Standards (NESTD) Std. DOE-NESTD-1004-92, 1992.

[24] E. Dijkstra, "The structure of the "THE"-multiprogramming system," *Communications of the ACM*, vol. 11, no. 5, pp. 341–346, 1968.

[25] N. Dulac and N. G. Leveson, "An approach to design for safety in complex systems," in *Proceedings of the* 14^{th} *Annual International Symposium on Systems Engineering (INCOSE'04)*. Toulousse, France: International Council on Systems Engineering, June 2004, pp. 33–407.

[26] M. S. Feather, "Language support for the specification and development of composite systems," *ACM Transactions on Programming Languages and Systems*, vol. 9, no. 2, pp. 198–234, Apr. 1987.

[27] M. S. Feather, S. Fickas, A. v. Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior," in *Proceedings of the* 9^{th} *International Workshop on Software Specifications and Design (IWSSD '98)*, Ise-Shima, Japan, Apr. 1998, pp. 50–59.

[28] B. W. Finnie, "Design for safety," in *Proceedings of the IEE Colloquium on Safety Critical Software in Vehicle and Traffic Control*, Feb. 1990.

[29] K. Forsberg and H. Mooz, "The relationship of system engineering to the project cycle," in *Proceedings of the* 1^{st} *Annual Conference of the National Council For Systems Engineering (NCOSE'91)*, Chattanooga, TN, USA, Oct. 1991.

[30] P. L. Goddard, "Software FMEA techniques," in *Proceedings of the Annual Reliability and Maintainablity Symposium*, Los Angeles, CA, USA, Jan. 2000, pp. 118–123.

[31] A. Hall, "Seven myths of formal methods," *IEEE Software*, vol. 7, no. 5, pp. 11–19, Sept. 1990.

[32] K. M. Hansen, A. P. Ravn, and V. Stavridou, "From safety analysis to software requirements," *IEEE Transactions on Software Engineering*, vol. 24, no. 7, pp. 573–584, Sept. 1998.

[33] A. L. Hopkins and T. B. Smith, "The architectural elements of a symmetric fault-tolerant multiprocessor," *IEEE Transactions on Computers*, vol. C-24, no. 5, pp. 498–505, May 1975.

[34] IABG, "V-model lifecycle process model: Brief description," Industrieanlagen-Betriebsgesellschaft mbH (IABG), Ottobrunn, Germany, Tech. Rep., Feb. 1993.

[35] *Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems*, International Electrotechnical Commission (IEC) Std. IEC61 508, 1997.

[36] M. Jackson and P. Zave, "Distributed feature composition: A virtual architecture for telecommunications services," *IEEE Transactions on Software Engineering*, vol. 24, no. 10, pp. 831–847, Oct. 1998.

[37] C. W. Johnson. (2003) Tutorial: Causal analysis for incident and accident investigation. Accessed on May 1, 2009. [Online]. Available: http://www.dcs.gla.ac.uk/advises/tutorials/chris_accidents.html

[38] C. Johnson and M. Bowell, "Using software development standards to analyse accidents involving electrical, electronic or programmable electronic systems: The blade mill case study," in *2nd Workshop on the Investigation and Reporting of Incidents and Accidents 2003*, C. C.J. Hayhurst and B. Strauch, Eds., no. NASA/CP-2003-212642. Virginia, USA: NASA Langley Research Centre, 2003, pp. 111–128.

[39] S. E. Keller, L. G. Kahn, and R. B. Panara, "Specifying software quality requirements with metrics," in *System and Software Requirements Engineering*, R. H. Thayer and M. Dorfman, Eds. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990, ch. 3, pp. 145–163.

[40] M. Kim, M. Viswanathan, H. Ben-Abdallah, S. Kannan, I. Lee, and O. Sokolsky, "Formally specified monitoring of temporal properties," in *Proceedings of the* 11^{th} *Euromicro Conference on Real-Time Sys. 1999*, York, UK, June 1999, pp. 114–122.

[41] P. Koopman, "A taxonomy of decomposition strategies based on structures, behaviors, and goals," in *Proceedings of the* 9^{th} *Conference on Design Theory and Methodology*, Boston, MA, USA, Sept. 1995, pp. 611–618.

[42] R. Koymans, Ed., *Specifying Message Passing and Time-Critical Systems With Temporal Logic*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 1992, vol. 651.

[43] A. v. Lamsweerde, "Requirements engineering in the year 00: a research perspective," in *Proceedings of the* 22^{nd} *International Conference on Software Engineering (ICSE00)*, Limerick, Ireland, June 2000, pp. 5–19.

[44] A. v. Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Proceedings of the* 5^{th} *IEEE International Symposium on Requirements Engineering (RE01)*, Toronto, Canada, Aug. 2001, pp. 27–31.

[45] A. v. Lamsweerde, R. Darimont, and E. Letier, "Managing conflicts in goal-driven requirements engineering," *IEEE Transactions on Software Engineering*, vol. 24, no. 11, pp. 908–926, Nov. 1998.

[46] E. Letier, "Reasoning about agents in goal-oriented requirements engineering," Ph.D. dissertation, Univ. catholique de Louvain, Louvain, Belgium, May 2001, accessed on April 6, 2009. [Online]. Available: http://www.info.ucl.ac.be/Research/Publication/Theses/letier.pdf

[47] E. Letier and A. v. Lamsweerde, "Agent-based tactics for goal-oriented requirements elaboration," in *Proceedings of the* 24^{th} *International Conference on Software Engineering (ICSE'02)*, Orlando, FL, USA, May 2002, pp. 83–93.

[48] E. Letier and A. v. Lamsweerde, "Deriving operational software specifications from system goals," in *Proceedings of the* 10^{th} *ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE-10)*, Charleston, SC, USA, Nov. 2002, pp. 119–128.

[49] E. Letier and A. v. Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," *SIGSOFT Software Engineering Notes*, vol. 29, no. 6, pp. 53–62, 2004.

[50] N. G. Leveson, *Safeware - System Safety and Computers*. Reading, MA, USA: Addison-Wesley, 1995.

[51] N. G. Leveson, "Intent specifications: An approach to building human-centered specifications," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 15–35, Jan. 2000.

[52] N. G. Leveson, "A new approach to hazard analysis for complex systems," in *Proceedings of the* 21^{st} *International System Safety Conference (ISSC'04)*. Ottawa, Canada: System Safety Society, Aug. 2003.

[53] N. G. Leveson, "Model-based analysis of socio-technical risk," Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep. ESD-WP-2004-08, Dec. 2004.

[54] N. G. Leveson, "A systems-theoretic approach to safety in software-intensive systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 66–86, Jan. 2004.

[55] N. G. Leveson and P. R. Harvey, "Analyzing software safety," *IEEE Transactions on Software Engineering*, vol. 9, no. 5, pp. 569–579, Sept. 1983.

[56] N. G. Leveson, T. J. Shimeall, J. L.Stolzy, and J. C. Thomas, "Design for safe software," in *Proceedings of the 21st AIAA Aerospace Sciences Meeting*, Reno, NV, USA, Jan. 1983.

[57] G. H. Lewes, *Problems of Life and Mind (First Series)*. Boston, MA, USA: James R. Osgood and Co., 1875, vol. 2.

[58] P. Li, L. Alvarez, and R. Horowitz, "AHS safe control laws for platoon leaders," *IEEE Transactions on Control Systems Technology*, vol. 5, no. 6, pp. 614–628, Nov. 1997.

[59] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems," in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA, USA, Jan. 1993, pp. 126–133.

[60] R. R. Lutz and C. Mikulski, "Empirical analysis of safety-critical anomalies during operations," *IEEE Transactions on Software Engineering*, vol. 30, no. 3, pp. 172–180, Mar. 2004.

[61] R. R. Lutz and H. Shaw, "Applying adaptive safety analysis techniques," in *Proceedings of the* 10^{th} *International Symposium on Software Reliability Engineering (ISSRE'99)*, Boca Raton, FL, USA, Nov. 1999, pp. 42–49.

[62] R. R. Lutz and R. M. Woodhouse, "Requirements analysis using forward and backward search," *Annals of Software Engineering*, vol. 3, pp. 459–475, Nov. 1997.

[63] R. R. Lutz and R. M. Woodhouse, "Bi-directional analysis for certification of safety-critical software," in *Proceedings of the* 1^{st} *International Software Assurance Certification Conference (ISACC'99)*, Washington, DC, USA, Mar. 1999.

[64] Mechanical Simulation, *VehicleSim SGUI (Simulation Graphical User Interface)Reference Manual*. Ann Arbor, MI, USA: Mechanical Simulation Corporation, 2007.

[65] J. Mylopoulos, L. K. Chung, and B. A. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, June 1992.

[66] N. J. Nilsson, *Problem-Solving in Artificial Intelligence*, ser. McGraw-Hill Computer Science Series, R. W. Hamming and E. A. Feigenbaum, Eds. New York, NY, USA: McGraw-Hill, 1971.

[67] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *The future of software engineering 2000 : 22nd International Conference on Software Engineering*, A. Finkelstein, Ed. New York, NY, USA: Association for Computing Machinery, 2000, ch. 10, pp. 35–46.

[68] D. K. Peters and D. L. Parnas, "Requirements-based monitors for real-time systems," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 146–158, Feb. 2002.

[69] B. Plale and K. Schwan, "Run-time detection in parallel and distributed systems: Application to safety-critical systems," in *Proceedings of the* 19^{th} *IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, Austin, TX, USA, June 1999, pp. 163–170.

[70] B. A. Plale, "Software approach to hazard detection using on-line analysis of safety constraints," Ph.D. dissertation, State University of New York at Binghamton, Binghamton, NY, USA, 1998, accessed on April 6, 2009. [Online]. Available: http://www.cs.indiana.edu/~plale/documents/Plale_PhDthesis.pdf

[71] J. Pollard and E. D. Sussman, "An examination of sudden acceleration," National Highway Traffic Safety Administration, Washington, DC, USA, Tech. Rep. DOT-HS-807-367, 1989.

[72] M. Privosnik, M. Marolt, A. Kavcic, and S. Divjak, "Evolutionary construction of emergent properties in multi-agent systems," in *Proceedings of the* 11^{th} *Mediterranean Electrotechnical Conference (MELECON'02)*, Cairo, Egypt, May 2002, pp. 327–330.

[73] *Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics (RTCA) Std. DO-178B, 1992.

[74] D. T. Ross, "Structured analysis (SA): a language for communicating ideas," *IEEE Transactions on Software Engineering*, vol. 3, no. 1, pp. 16–34, Jan. 1977.

[75] D. T. Ross and K. E. S. JR., "Structured analysis for requirements definition," *IEEE Transactions on Software Engineering*, vol. 3, no. 1, pp. 6–15, Jan. 1977.

[76] J. Rushby, "Kernels for safety?" in *Safe and Secure Computing Systems*, T. Anderson, Ed. Blackwell Scientific Publications, 1989, ch. 13, pp. 210–220.

[77] W. D. Salyer and D. J. Bizzak. (2002) An overview of research and findings in the investigation of sudden acceleration incidents involving model year 1991 through 1995 jeep cherokee and grand cherokees. Accessed on April 6, 2009. [Online]. Available: www.rdaweb.com/jeepsa/NHTSA%20Defect%20Petition.pdf

[78] B. Schroeder, K. Schwan, and S. Aggarwal, "Software approach to hazard detection using on-line analysis of safety constraints," in *Proceedings of the* 16^{th} *Symposium on Reliable Dist. Sys. (SRDS'97)*, Durham, NC, USA, Oct. 1997, pp. 80–87.

[79] O. Sokolsky, U. Sammapun, I. Lee, and J. Kim, "Run-time checking of dynamic properties," in *Proceedings of the* 5^{th} *Workshop on Runtime Verification*, Edinburgh, UK, July 2005.

[80] N. Storey, *Safety-Critical Computer Systems*. Reading, MA, USA: Addison-Wesley Publishing Company, 1996.

[81] C. Temple, "Avoiding the babbling-idiot failure in a time-triggered communication system," in *Proceedings of the* 28^{th} *International Symposium on Fault-Tolerant Computing (FTCS-28)*, Munich, Germany, June 1998, pp. 218–227.

[82] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematics Society*, vol. 42, pp. 230–265, 1936.

[83] *Procedures for Performing a Failure Mode, Effects and Criticality Analysis*, U.S. Department of Defense Std. MIL-STD-1629A, 1980.

[84] H. Velthuijsen, "Distributed artificial intelligence for runtime feature-interaction resolution," *IEEE Computer*, vol. 26, no. 8, pp. 48–55, Aug. 1993.

[85] W. E. Veseley, F. F. Goldberg, N. H. Roberts, and D. F. Haasl., *Fault Tree Handbook*, U.S. Nuclear Regulatory Commission Std. NUREG-0492, 1981.

[86] K. G. Wika, "Safety kernel enforcement of software safety policies," Ph.D. dissertation, University of Virginia, Charlottesville, VA, USA, May 1995, accessed on April 6, 2009. [Online]. Available: citeseer.ist.psu.edu/wika95safety.html

[87] K. G. Wika and J. C. Knight, "On the enforcement of software safety policies," in *Proceedings of the* 10^{th} *Annual Conference on Computer Assurance (COMPASS '95, 'System Integrity, Software Safety and Process Security')*, Gaithersburg, MD, USA, June 1997, pp. 83–93.

[88] P. Zave, "Feature interactions and formal specifications in telecommunications," *IEEE Computer*, vol. 26, no. 8, pp. 20–29, Aug. 1993.