Chapter 4: Extending STAMP and STPA for Collaborative Control

The system-theoretic foundation of STAMP and STPA is well-suited to address some of the challenges in modeling and analyzing the novel aerospace systems introduced in Chapter 1.1. However, these methods need additional guidance to clarify how to handle causality associated with the more complex team-inspired component interactions sought in new designs. While STPA may find some causal factors associated with collaborative control, it is vulnerable to missing others because it lacks a systematic approach to address such relationships.

This problem was evident when analyzing the safety of a future helicopter concept that executes missions *optionally-manned*[5] and in collaboration with multiple UAS [28]. The system was modeled with a control structure that includes a "teaming controller" to help STPA explore teaming interactions (Figure 4-1). The function of the teaming controller is to dynamically coordinate resources in the multi-aircraft team to address mission needs. The teaming controller could be implemented as a centralized, distributed, or hybrid controller.

The model did help identify several causal scenarios that resulted from the breakdown of collaborative control among the aircraft. The results include examples of conflicting control, unsafe control handoffs, human-machine trust issues, inconsistent semantics in the team, incompatible system configurations, and more [28]. While these results are good and useful, several challenges were encountered during the analysis.

For instance, the causal relationships for many of the unsafe collaborative control issues listed above are not expressed in the control structure. The results were only obtained by analyzing the model creatively, through the lens of teaming, and drawing on past expertise in collaborative control. Earlier versions of the analysis, without that perspective, did not consider the issues listed above. For a technique to be repeatable by different analysis teams, the causal influences that occur in collaboration must be explicitly accounted for by the model and analyzable using a systematic process.

The teaming controller also led to ambiguity in the hierarchy of components in the model. For example, by controlling team resources, which include the operator's aircraft, the teaming controller may issue control actions to the operator and therefore be hierarchically superior. However, the reverse also occurs when the operator directs the actions for the team, as represented in the control structure.

Challenges also occurred in abstracting the model to a higher level. It is ambiguous if the teaming controller belongs to the operator, the aircraft software controller, the UAS, or all three

[5] *Optionally-manned* means a human pilot may or may not be physically onboard the aircraft. If no pilot is onboard, the aircraft receives piloting commands remotely.

if distributed. Similarly, it is unclear how a higher-level model of such a distributed system would be consistently refined to produce the representation in Figure 4-1.

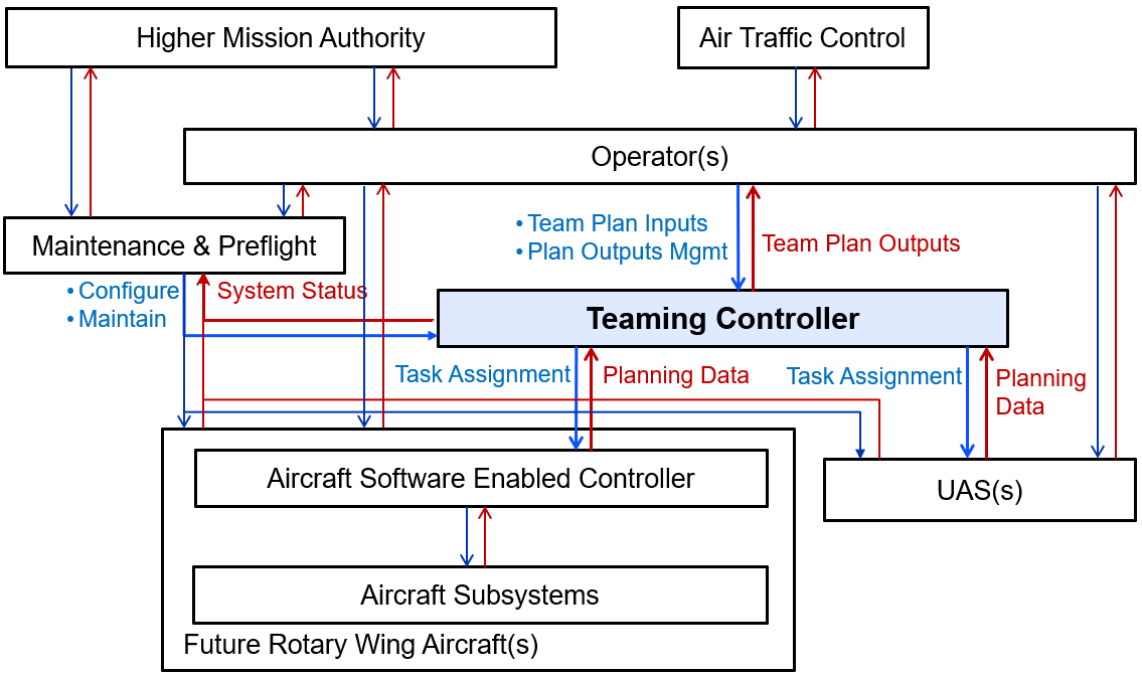

Figure 4-1: "Teaming Controller" in Future Helicopter Control Structure (adapted from [28])

Despite the useful results obtained with the teaming controller as represented, the lack of a systematic approach to handle collaborative control could have resulted in the system being modeled differently. This variation can lead to different results in the hazard analysis and increases the risk of missing causal factors. As such, further guidance is needed in STAMP and STPA to handle these situations more consistently.

This chapter introduces extensions to STAMP and STPA to add guidance and rigor in the analysis of collaborative control interactions. The new methods are grounded in the collaborative control framework defined in Chapter 3. They are also derived from the existing guidance in STAMP and STPA so that they remain consistent with these proven techniques. The following is an overview of the three extensions, which are illustrated in Figure 4-2 and are collectively referred to as *STPA-Teaming*.

First, a *generic collaborative control structure* is developed to incorporate the types of interactions exhibited in teaming into STAMP models. It serves as a reconfigurable template to assist in modeling the relationships between multiple humans and/or machines that collaborate in the control of a process. The goal is to explicitly express these mutual influences so that causal factors associated with them can later be systematically identified.

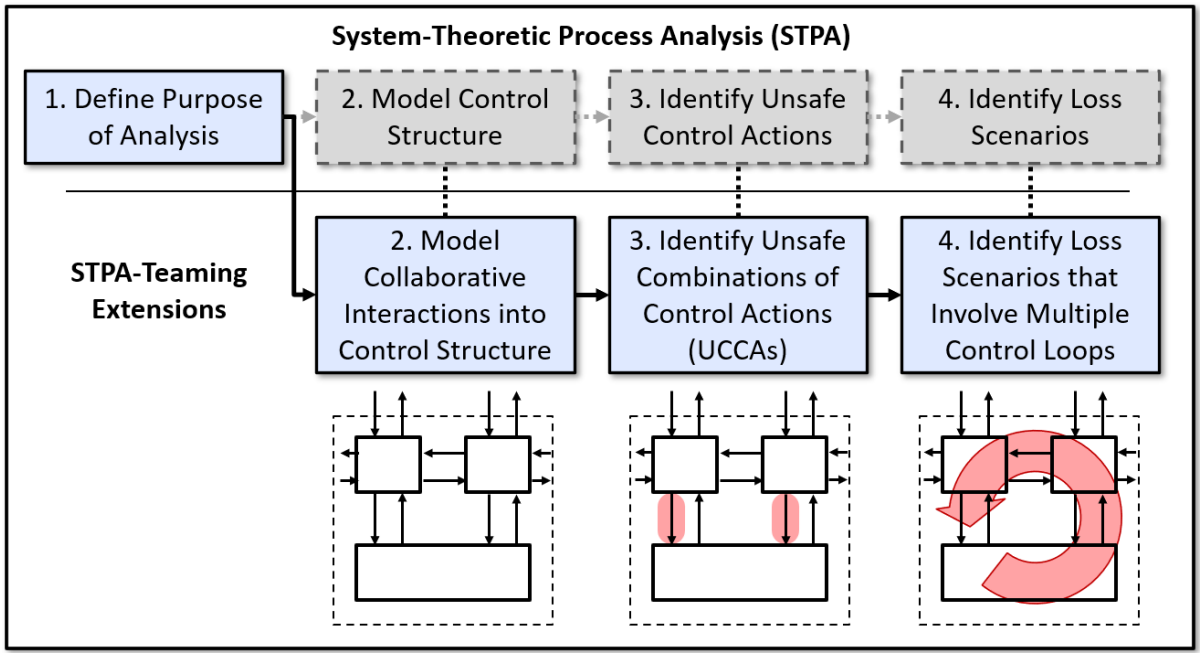

Figure 4-2. Three Analytical Extensions Involved in STPA-Teaming

Second, the process to identify unsafe control actions (UCAs) in STPA is expanded to explore how combinations of multiple control actions provided together may lead to hazards. The actions of multiple controllers are systematically analyzed relative to one another using an approach derived from the four *types of UCAs* defined in STPA [50]. A method of abstracting and refining the control structure helps manage combinatorial complexity in this process. Finally, a prototype automation tool is introduced to support an analyst with enumerations, refinement, and prioritization of the unsafe combinations of control actions to analyze. The overall procedure systematically considers potential issues involving control gaps, overlaps, transfers, and controller-task mismatches that are found in collaborative control.

Third, an analytical procedure is introduced to develop causal scenarios that explain how these unsafe combinations of control actions (UCCAs) could occur. It follows a structured search process inspired by Thomas [203] and is framed by the collaborative control dynamics defined in Chapter 3. The method emphasizes defining scenarios at a high level and refining them iteratively, as necessary, and guided by the types of interactions between controllers.

The remainder of the Chapter provides details associated with the development of each of these extensions. These techniques are then demonstrated in a case study on a real-world aerospace system concept in Chapter 5.

4.1 Generic Collaborative Control Structure

STAMP hierarchal control structures provide a powerful mechanism to model complex systems holistically and top-down. The ability to represent causal relationships between components at multiple levels of the system is one of the reasons why STAMP is so successful at finding factors

that threaten safety. In recent years, several efforts have aimed to enhance STAMP models so that they produce more complete analyses.

The synthesis of these efforts helped define the generic control structure shown in Figure 4-3 and found in Appendix G of the STPA Handbook [50]. This reference also explains how each element of the model can contribute to causal scenarios found using STPA. The model consists of a human operator that supervises automation, which controls a process. It is representative of many supervisory control systems currently fielded.

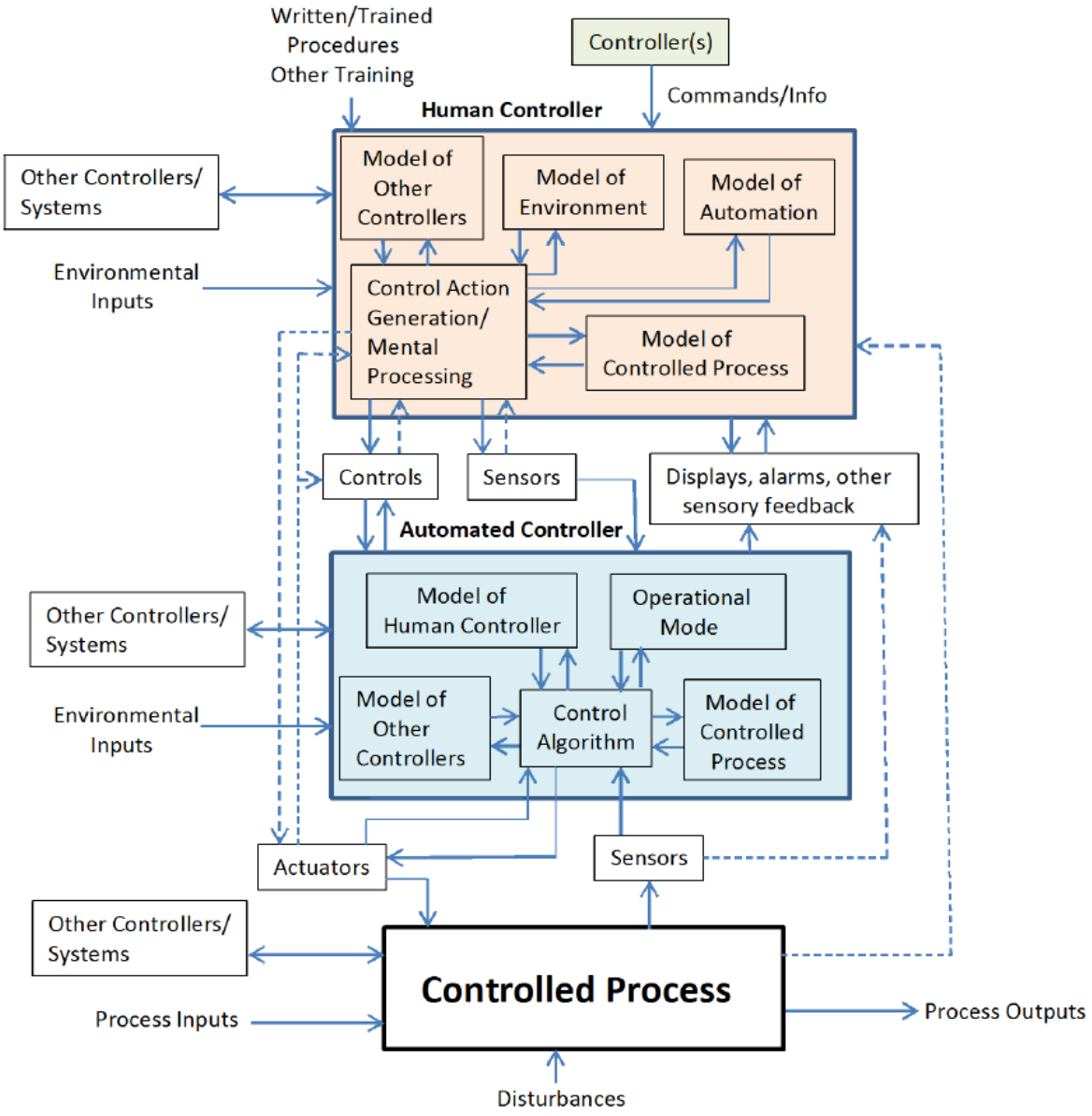

Figure 4-3. Baseline Generic Control Structure from STPA Handbook Appendix G [50]

Unfortunately, the collaborative control dynamics defined in Chapter 3 are not explicitly represented in this model, nor in other existing control structures. This increases the risk that those causal relationships will either be missed or will not be systematically handled in the ensuing hazard analysis. To address this gap, this dissertation introduces the *Generic Collaborative*

Control Structure shown in Figure 4-4 as a reconfigurable template to represent various teaming systems. The model is grounded in STAMP, but it also extends the available guidance to incorporate collaborative interactions.

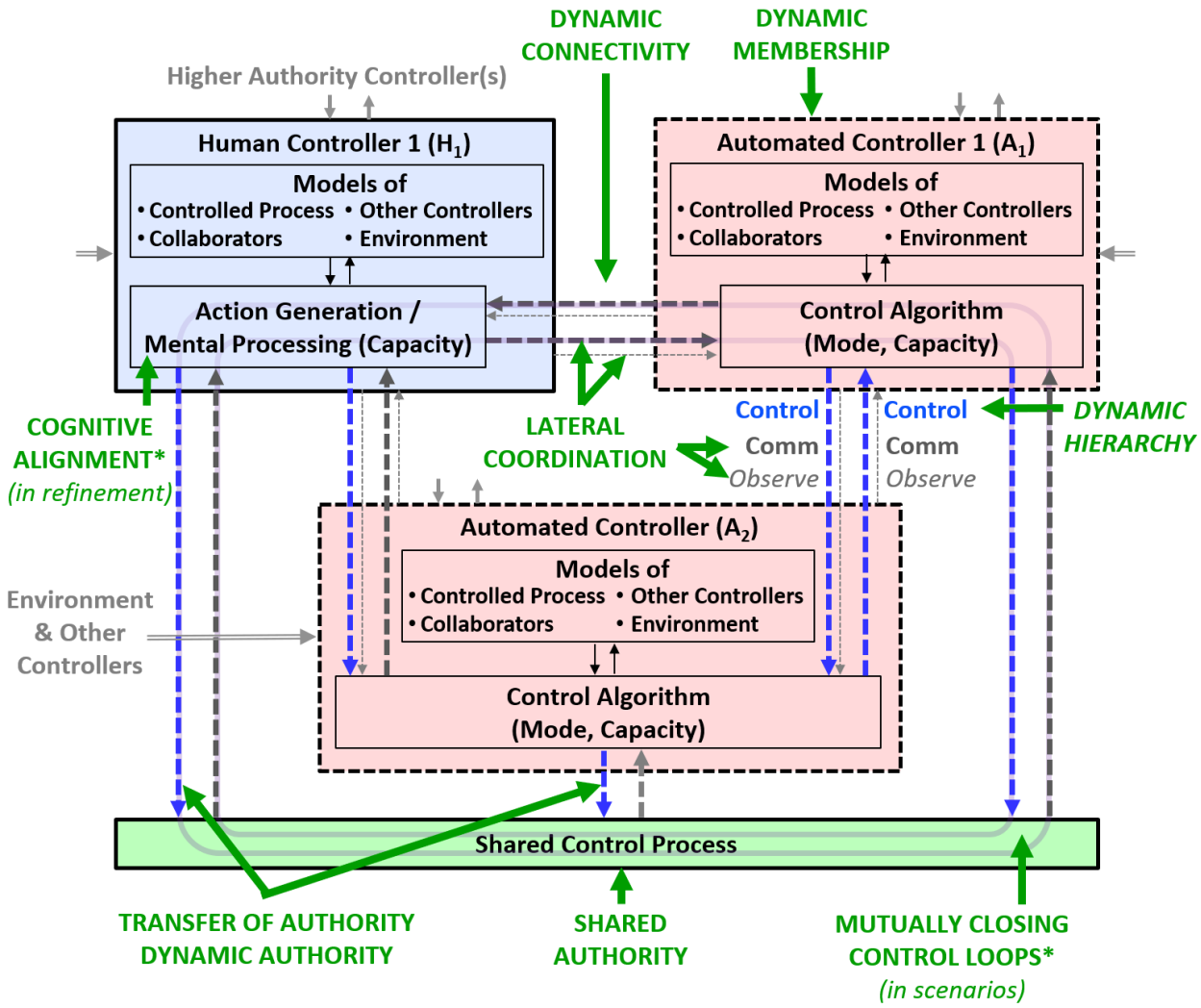

Figure 4-4. Generic Collaborative Control Structure

The remainder of this section is organized as follows. Section 4.1.1 provides an overview of the model and relates it to STAMP. Section 4.1.2 describes the cognitive functions of the controllers in the model, which underpin several of the collaborative control dynamics. Section 4.1.3 explains how the collaborative control dynamics are integrated into the control structure. Finally, Section 4.1.4 provides additional modeling recommendations.

4.1.1 Overview of the Model

The generic collaborative control structure builds on the baseline in Figure 4-3 to express collaborative interactions while remaining consistent with STAMP. Every element described in the baseline applies to the extension. However, abstraction allows some of the details to be hidden so that other features more aligned with the research focus on collaboration can be

highlighted. For example, the *actuators* in Figure 4-3 are abstracted away from the control paths, as are the *sensors* in the feedback paths. These lower-level components are still part of the collaborative control model, and if needed can be handled using the guidance from the baseline.

Each controller in the extended model includes the same high-level cognitive functions described in the baseline. The relationship of these functions to collaborative control is further discussed in Section 4.1.2. The interactions the controllers have with higher authority controllers, with other controllers (beyond the set of collaborators), and with the environment are also shown and are handled no differently than they are in STPA.

The generic collaborative control structure shown in Figure 4-4 represents one of any arbitrary collaborative system configurations. In this case, it includes a human controller (H_1) working as a peer with an automated controller (A_1). Together, H_1 and A_1 have authority over another automated controller (A_2). All three controllers collaborate in controlling a shared process.

These building blocks can be reorganized into any other configurations. For instance, Figure 4-5 shows the collaborative control structures of different systems in development for Urban Air Mobility (UAM). These concepts involve different potential architectures for human and/or automated controller collaborations [8], [10], [204]. The *authority bus* in the figure indicates shared authority by the controllers over the subprocesses and is further discussed in Section 4.1.4.

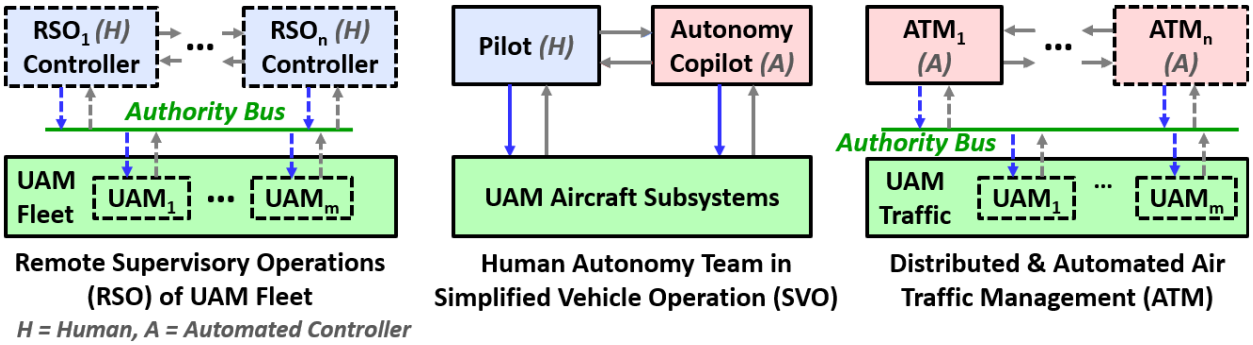

Figure 4-5. System Concepts Demonstrating Various Collaborative Control Configurations

The system-theoretic framework allows collaboration to be represented at multiple levels of abstraction and hierarchy. For example, while Figure 4-4 shows all three controllers collaborating to control the shared process, it can also be analyzed as H_1 and A_1 collaborating in the shared control of A_2. Similarly, the human-machine system modeled could be abstracted as a whole, and work collaboratively with other human-machine systems on a shared process. The interactions highlighted in the collaborative control structure can be applied to any set of collaborators sharing a process. Strategies to navigate between these different views are presented in Chapter 6.

4.1.2 Cognitive Functions

The baseline causal model (Figure 4-3) defines two sets of high-level cognitive functions for each controller [50]. The first processes information to make control decisions. The second consists of various models that support the decision-making process. STPA provides guidance to consider how a controller may have flaws in these functions that contribute to its unsafe control actions.

The collaborative control structure carries over these concepts, but it emphasizes instead how these functions can be flawed relative to one another across multiple collaborating controllers. Figure 4-6 provides an overview of how cognitive functions are integrated into the extended model. The following discussion explains these functions and how they were derived from the baseline guidance and other references. These processes are highly relevant to many of the collaborative control interactions further discussed in Section 4.1.3.

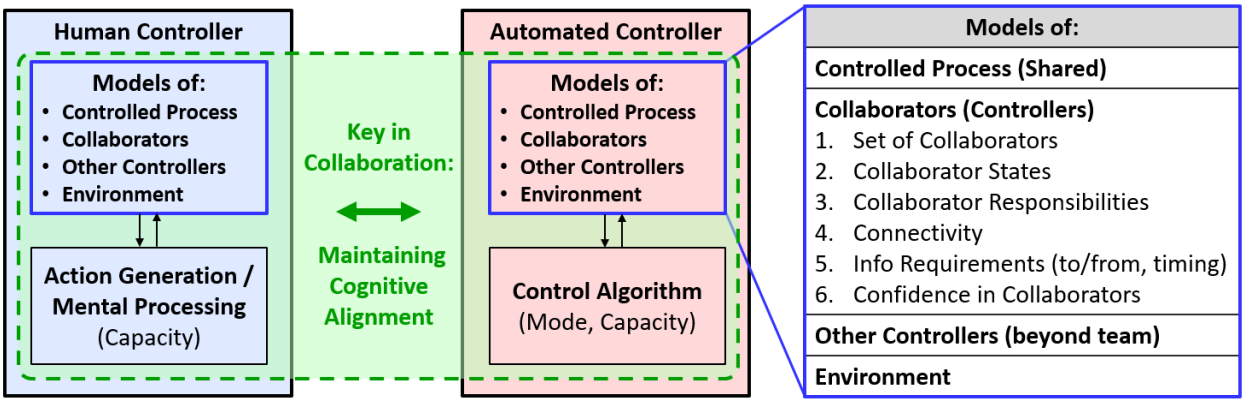

Figure 4-6. Cognitive Functions in Collaborative Control Structure

Information Processing and Decision Making

As explained in the baseline, automated controllers have a *control algorithm* that processes inputs to the controller to (1) generate control actions, and (2) maintain accurate information about the state of the system by interacting with the controller models. The behavior of the control algorithm is shaped by the operational modes of the system [50].

The extended model integrates all these components. One subtle difference is that the control algorithm is emphasized to generate *actions*, which include *control actions* as in the baseline, but it also consists of other *communication actions* to coordinate with and influence the behavior of other controllers without using control. For example, the Aircraft Collision Avoidance System (ACAS-X) system described in Chapter 3.3 allows the control algorithm of a lower-ranked aircraft to output a deconfliction solution first, which influences the decision of the higher-ranked aircraft.

Another consideration added to the model is the limitations placed on the control algorithm due to controller *capacity*. In Johnson's work, limited capacity is a key factor in determining how controllers form dependencies on one another to execute joint activities [2]. He broadly defines capacity as the set of knowledge, skills, abilities, and resources a controller needs to perform an action. Some of these elements are already explicitly considered in STAMP in the form of control paths, feedback paths, and models. However, other aspects require attention and are important in collaborative interactions.

In this work, the capacity of a control algorithm refers to the factors that can limit its ability to track and update models and select appropriate output actions. Limited internal resources, including computational, data, or communication, may prevent a controller under a certain workload to output the behavior that was expected by its collaborators. Such issues can cause a misalignment in cognition across multiple controllers and contribute to an unsafe team output.

These concepts also apply to the human controllers in the extended model. The baseline represents the human with a cognitive function for *control action generation & mental processing* [50]. While its purpose is comparable to that of the *control algorithm* in machines, the STPA guidance emphasizes that humans are more complex and are subject to different causal factors.

The collaborative control structure similarly broadens the scope of the function to *action generation & mental processing* so that it accounts for the coordination outputs in addition to the control actions. Humans working collaboratively often deliberately provide coordination actions to influence the behavior of others they cannot control. For example, flight crews often verbalize hints to each other regarding recommended actions.

Similarly, humans are also subject to *capacity* limitations in their ability, skill, or workload that influence how well they maintain situational awareness and make decisions. For instance, an inexperienced pilot who is still learning to process information efficiently will more easily be overwhelmed, "fall behind the airplane", and make bad decisions.

Model of the Process

The cognitive function described above updates, maintains, and relies on multiple models of the system to select the actions to generate [50]. The purpose of the models is similar between humans and automated controllers, but the reasons for their flaws can vary greatly. As with other baseline STPA elements, these models are integrated into the generic collaborative control structure and related to the process of collaboration.

In STAMP, the model of the process represents the state that the controller believes the process to be in. The controller relies on this model to select control actions that will constrain the behavior of the process so that it does not enter a hazardous state [38]. The STPA guidance provides many reasons why an automated controller may have a flawed process model. These include inadequate feedback from the process, delays in receiving or processing the feedback, and flawed assumptions based on control inputs. Human controllers may, in addition, be subject to mode confusion, lack of situational awareness, confusion due to lack of transparency, or even complacency [50]. These all apply to the collaborative control structure.

However, collaborative control brings on additional considerations as the *control process is shared* between multiple controllers. It is not sufficient to just consider what state one controller believes the process to be in. The consistency of the process models across multiple controllers is critical. As reviewed in Chapter 2.1, teams rely on both shared and distributed cognition.

Shared cognition relates to information held in common between multiple controllers [60]. If information is misaligned, the controllers may have different beliefs regarding the process state and may issue commands that are inconsistent with one another. This was likely a contributing factor in the Air France Flight 447 accident (see Chapter 1.2), when the two pilots had misaligned models about the state of the aircraft, leading one to pitch up and the other to pitch down [35].

Shared cognition may also cause a controller that has a valid model of the process to drift toward incorrect beliefs provided by a collaborator. This can occur in the psychological phenomenon of *groupthink*. In addition, a controller that has a flawed process model may receive negative reinforcement from the flawed model of a collaborator. This can also occur in *groupthink* and in *confirmation bias*. As such, the model flaws in one controller can propagate to others.

Distributed cognition focuses on differences in knowledge between collaborators and emphasizes the need to coordinate around who knows what and when on the team. The key implication of this concept is that in collaboration, any one controller may not have direct access to the complete state of the process to decide what actions to take. It may rely on other controllers to receive the necessary feedback to inform its actions.

Flawed distributed cognition contributed to the 1994 friendly fire shootdown of two U.S. Army helicopters (see Chapter 1.2). Two combat air traffic controllers had split responsibilities to respectively maintain mental models of aircraft within and outside a prescribed area. The responsibility to track low-flying helicopters evolved over time and was eventually left unassigned for particular situations. In the accident, the controller for inside the area relied on an incomplete model from the other controller and, as a result, made an unsafe decision [38].

Model of the Collaborating Controllers

The STPA guidance also describes how a controller may have a model of other controllers it interacts with. For example, a human controller must have a model of the automation to supervise its control of the process. Similarly, some sophisticated automated controllers have models of the humans that are supervising them [50]. In this work, the generic collaborative control structure incorporates this concept as a *model of collaborators*.

Several information items may form the model of collaborators. Examples found in the systems sampled in Appendix 1 are illustrated on the right of Figure 4-6. While the content of these models can vary widely, the intent of the figure is to help reason about the type of information controllers track about their teammates to shape their individual output decisions.

The model of collaborators may include knowledge of the set of collaborators involved in the activity. This becomes particularly important in teams that exhibit dynamic membership when this set changes over time. A controller may exhibit unsafe behavior if it is not aware that it has a teammate or if it falsely believes that it does.

Some systems require controllers to track the state of their collaborators, such as their location and trajectory. For example, in implicit coordination algorithms, each controller uses the state information for all members of the team to compute a plan for the whole team and execute their portion of the plan. If the state information and the algorithms are consistent across controllers, they can produce safe, coordinated solutions [83].

In some cases, teammates may be required to track the responsibilities of other collaborators. Such information can be necessary for collaborative systems that exhibit dynamic authority, in which controllers determine the allocation of control during execution. In some coordination schemes, such as market-based algorithms, teammates form consensus over their responsibilities only and do not have to rely on other state information [83].

Controllers on a team may need to estimate the network topology. Many systems exhibit dynamic connectivity, in which communication channels between controllers are expected to vary. The knowledge of which controllers can be reached at any time, either directly or indirectly, may influence how a control decides to output coordination and control actions.

Beyond connectivity, controllers may track information requirements between controllers. In distributed cognition, controllers must understand what information they need from others and what information others need from them [39]. The timing requirements for information sharing

may also vary based on the context and the types of controllers involved. There are many examples of automated controllers that unduly interrupt the workflow of humans, or that update information too quickly for humans to process [23].

Finally, controllers may assess their confidence in the behavior of other controllers. A controller that has no confidence in the output of a teammate may choose a different action than if it did. Asymmetric assessments of confidence lead to misaligned decisions in joint control.

Models of Other Controllers and the Environment

As described for STPA, controllers can rely on additional models in selecting their actions. They may have a *model of other controllers* involved in the system. In this work, these are controllers beyond the set of *collaborative controllers* considered above. These other controllers may interact with the team, the process, or some other part of the system in a different way. Similarly, the controllers may also maintain models of the environment, or components beyond the system boundary that the system interacts with. In this work, these models are as they are in STPA [50].

4.1.3 Collaborative Control Dynamics in the Control Structure

A key goal of the extended control structure is to express the collaborative control dynamics defined in Chapter 3 that are exhibited by the system. This section explains how this is accomplished using the items labeled with the green arrows in Figure 4-4.

A key consideration in collaborative control is *shared authority*. It is expressed in the control structure with the *shared controlled process*, which is the joint control activity over which multiple controllers have authority. It may represent a set of mission tasks, a formation, trajectory deconfliction, or any other process that is jointly controlled.

Many past STPA studies relevant to teaming, including the one referenced in Figure 4-1, do not show this process in the control structure, and instead, only list the various controllers that collaborate. The inclusion of a shared process is necessary to reason about how combinations of control actions by multiple controllers may be hazardous. This concept is central to the Unsafe Combination of Control Action (UCCA) identification technique introduced in Section 4.2.

The use of dashed lines in the collaborative control structure symbolizes the dynamic presence of an item. This convention is repeated in multiple ways. The dashed control and feedback arrows that lead to and from the shared process indicate *dynamic authority* or *transfer of authority*. The significance in both is that the controller from which the dashed control line originates may not always be responsible for issuing the control action.

In dynamic authority, the control path is allocated, or possibly reallocated multiple times, to one of the collaborative controllers during execution. In transfer of authority, the control path is handed off from one controller to another over time. Control gaps, overlaps, and mismatches that can arise from these dynamics may lead the system into a hazardous state. These two concepts are also inherently captured by the UCCA extension described in Section 4.2.

The arrows between the controllers are dashed to symbolize *dynamic connectivity*. Those connections may or may not be available at any given time. Similarly, some of the controllers have dashed frames to indicate *dynamic membership*. This means that those controllers are not

always part of the control structure. The causal implications of dynamic connectivity and dynamic membership, as well as those for all the remaining collaborative control interactions, are covered in the scenario identification process in Section 4.3.

In STPA, items listed on downward arrows from one controller to another are typically treated as control actions and analyzed for UCAs. The arrows pointing up are feedback items and those connecting controllers laterally consist of other information [50]. This convention is relaxed in the extended control structure to account for additional interactions in collaboration. While the control structure is still organized hierarchically, not every item flowing down is necessarily a control action, and control actions may be included on lateral and upward arrows.

In Figure 4-4, each connection from one controller to another includes two arrows. One arrow, shown in bold, originates from the *action generation* cognitive function. The other arrow, not bold, stems instead from the overall controller frame. Both arrows terminate at the cognitive function of a receiving controller as inputs to inform its decisions.

The bold arrow flowing out of the cognitive function reflects information deliberately provided by the controller to influence another controller. It consists of control actions (bold and blue) and communication actions to enable lateral coordination (bold not blue). To clarify authority, if the arrow has a control action, then it is shown in blue even if it also includes communication items. In the ensuing hazard analysis, control actions are analyzed for UCCAs (see Section 4.2) and communication actions are considered in causal scenario development (see Section 4.3).

The thinner arrow, which does not originate from the cognitive function, represents the information obtained by observing a controller, as part of lateral coordination. As described in Chapter 3.2, even though this information is not deliberately provided to coordinate, it can implicitly influence the decisions of collaborators. Observation items are also considered in the scenario development process (see Section 4.3).

In Figure 4-4, both A_1 and A_2 can provide control actions to one another. This is indicative of *dynamic hierarchy*, in which a controller leads part of the interaction, and another controller leads in another part. This dynamic is also captured in the scenario development process.

The extended model shows how *mutually closing control loops* can be identified between multiple controllers and the shared process. However, this dynamic is further explored in scenario development using a more focused control structure for the control loops being analyzed. Similarly, a label for cognitive alignment is also shown in Figure 4-4, but it is more closely considered by accounting for the cognitive functions of multiple controllers in scenario development. These topics are detailed in Section 4.3.

4.1.4 Additional Recommendations for the Model

The following additional recommendations can help analysts model collaborative systems. In some cases, it may be easier to include an *authority bus* to represent shared authority over multiple processes, as shown in Figure 4-5. The bus indicates that all the controllers that feed into it can have authority over the processes that receive an output from it.

In addition, an indexing scheme of $\{1, \ldots, n\}$ controllers, also shown in Figure 4-5, helps account for a variable number of similar types of controllers. In such cases, it is recommended that at least two controllers be shown so that the interactions between them can be expressed and considered in causal analysis. Including more than two similar controllers is often not necessary and adds complexity to the analysis.

4.2 Unsafe Combinations of Control Actions (UCCAs)

STPA employs a systematic method derived from Control Theory to identify unsafe control actions (UCAs). The method has been shown to be complete in its ability to describe how a particular command may be provided in an unsafe way [50], [205]. However, this process often lends itself to considering one controller and one of its feedback control loops at a time.

Collaborative control fundamentally involves multiple controllers working together to control a shared process. A key implication of *shared authority*, as defined in Chapter 3, is that the control actions from collaborating controllers cannot be analyzed individually. There may be unsafe causal factors that can only be identified when the actions of these multiple controllers are analyzed together.

For example, a flight crew may involve a pilot that controls aircraft attitude and trajectory using the flight control yoke and throttle, and a second pilot that controls aircraft configuration by selecting flaps and landing gear settings. Configuration changes alter the aerodynamic properties of the aircraft and therefore alter how attitude and trajectory are controlled. Similarly, variations in attitude and trajectory may cause the aircraft to enter an operating state that necessitates a configuration change. The control actions of each controller may be unsafe in the context of the actions provided by the other controller.

Collaborative control may also involve *dynamic authority* that enables multiple controllers to adjust task allocation during execution (see Chapter 3). However, that may result in control gaps, conflict in overlaps, or controller-task mismatches. Similarly, a system may allow *transfer of authority,* which can lead to unsafe control handoffs. These types of relationships were exhibited by the operator teams involved in the infamous Air France 447 and 1994 Friendly Fire accidents described in Chapter 1.

In STPA, a human analyst specifies the context in which a control action is unsafe [50]. Suggestions are provided in the STPA Handbook for how to do this. Thomas provides formalism to identify context using process variables derived from the system hazards [205]. However, these methods do not explicitly define how to explore a control action that may be unsafe relative to other commands.

Placke's work provides a useful first step to address this gap. His approach explores how one controller may interfere with another [206]. Specifically, he defines *Conflict UCAs* of the form: <*Controller 1 provides Command A*> prior to <*Controller 2 provides Command B*> violates <*design assumptions*>. However, the method does not address different types of multi-controller interactions, such as control gaps, unsafe handoffs, and dynamic tasking. Furthermore, the analysis is limited to pairwise comparisons of two control actions. Finally, the formulation only

relates to a subset of the four Types of UCAs defined in STPA. This means that there are other ways to issue control actions relative to one another that must be considered.

This section introduces an extension to STPA to explore how multiple control actions may be unsafe together. The process identifies Unsafe Combinations of Control Actions (UCCAs) and has the following attributes. First, it follows a systematic approach derived from STPA to ensure all relevant types of control combinations are considered. Second, it leverages multiple levels of abstraction to manage combinatorial complexity. And finally, the UCCAs enable the analysis of their causal factors to be framed by the collaborative control dynamics defined in Chapter 3.

The remainder of this section is organized as follows. Section 4.2.1 derives the different types of UCCAs and provides the foundation necessary to enumerate them algorithmically. Section 4.2.2 introduces an initial framework of abstraction to reduce the combinatorial complexity of the enumeration. Section 4.2.3 adds further abstraction to linearize the growth of UCCAs for more complex systems. Finally, Section 4.2.4 integrates these concepts into an algorithm to identify, refine, and prioritize UCCAs for their follow-on causal analysis. A prototype tool built on the formalism of this algorithm supports its execution by automating some of the steps.

4.2.1 Types of UCCAs

One of the strengths of STPA is its systematic process of considering how a control action, given a particular context, can lead to a hazardous state. UCAs are identified using the specific structure shown in Figure 4-7 [50].

	1	2	3	4	5
UCA Structure:	<Controller>	**<UCA Type>**	<Control Action>	<*Context*>	*[H]*

Figure 4-7. Structure of a UCA in STPA

The controller (item 1) and the control action (item 3) are obtained from the control structure. STPA defines four types of UCAs (item 2), listed below, in which a control action may be unsafe. The set of four types is provably complete to describe a given control effort [50].

- UCA Type 1: not providing the control action
- UCA Type 2: providing the control action
- UCA Type 3: providing the control action too early or too late
- UCA Type 4: providing the control action for too long or stopping it too soon

The structure in Figure 4-7 allows items 1-3 to be machine enumerated. This reduces the burden on a human analyst, who can instead focus on determining if there is a context for the UCA (item 4), and, if so, trace it to the hazard(s) it leads to (item 5). Examples of each type of UCA are provided in Table 2-3.

The process to identify Unsafe Combinations of Control Actions (UCCAs) builds on the approach for UCAs. The UCA structure is expanded to incorporate combinations of control actions. Different types of UCCAs are derived from the types of UCAs to maintain the rigor provided by STPA. The resulting formulation provides the foundation necessary to enumerate combinations of control actions algorithmically. As such, the potential UCCAs can be machine-

generated and, again, focus the human analyst on specifying the context in which the control combinations are unsafe.

Two UCCA types are defined. The first, UCCA Type 1-2, combines UCA Types 1 and 2, which describe whether or not a control action is provided at all. Type 1-2 UCCAs help find contexts in which providing none, some, or all of multiple control actions is unsafe. As a simple example, consider two controllers c_i and c_j, that can each provide control action u_a and u_b respectively. Potential Type 1-2 UCCAs can be enumerated as follows, using an extended UCA structure with the same color coding as in Figure 4-7:

Type 1-2 UCCA Example:

1. c_i **does not provide** u_a and c_j **does not provide** u_b when… *[H]*
2. c_i **does not provide** u_a and c_j **provides** u_b when… *[H]*
3. c_i **provides** u_a and c_j **does not provide** u_b when… *[H]*
4. c_i **provides** u_a and c_j **provides** u_b when… *[H]*

Second, the Type 3-4 UCCA combines UCA Types 3 and 4, which assume that the control action is provided, and instead focuses on the temporal sequence in which it occurs. Specifically, a Type 3 UCA considers when the control effort starts, rising from OFF to ON, as shown with a step function in Figure 4-8. A Type 4 UCA accounts for when the control effort ends, falling from ON to OFF. For a discrete command not provided over time, only the rising edge is considered, and Type 4 UCAs do not apply [50].

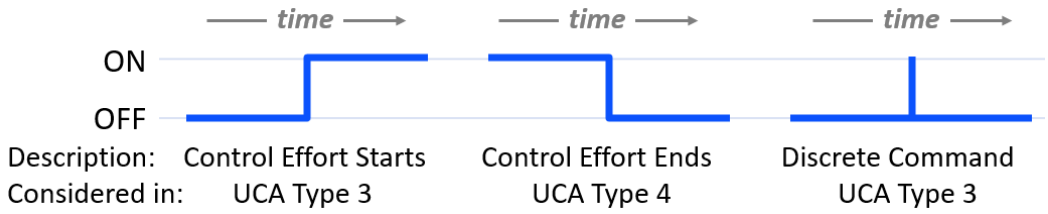

Figure 4-8. Start and End of a Control Effort Considered in UCA Type 3 and 4

As such, Type 3-4 UCCAs explore the temporal sequences of multiple control actions in relation to each other that are unsafe. Specifically, they analyze how starting or ending certain control actions before or after starting or ending other control actions may be unsafe. The Type 3-4 UCCAs are enumerated using the extended UCA structure in Figure 4-9 for the same example as above.

To maintain generality, no assumption is made that a control action, if started first, has not ended before the second action starts or ends. This subtlety accounts for discrete commands, which are not applied over time, as shown in Figure 4-9. The same assumption also allows the first command, if continuous, to be started and ended before the second action starts or ends. Similarly, the first command can be ended and then started again before the second command changes.

To remain consistent with UCA Type 4 in STPA, the end of a discrete command is not considered[6]. As such, if u_a is discrete, enumerations {3,4,7,8} in Figure 4-9 are not applicable. Similarly, if u_b is discrete, enumerations {2,4,6,8} are excluded. If both are discrete, only items {1,5} apply.

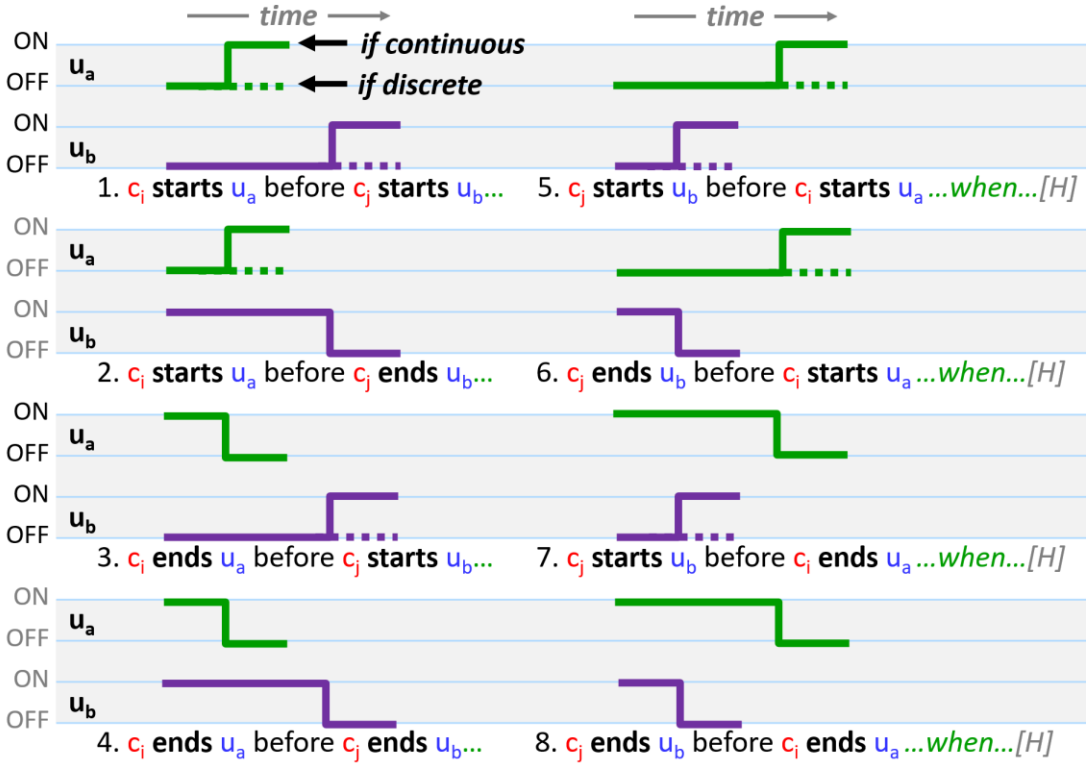

Figure 4-9. Type 3-4 UCCA Example

The two types of UCCAs provide the foundation for a machine to enumerate all possible control combinations. However, that does not by itself solve the problem of analyzing how multiple control actions are unsafe together. The example above consists of a simple pairwise comparison of two control actions. Additional processes, introduced in the next subsections, are needed to handle more complex combinations.

4.2.2 Managing Combinatorial Complexity with Abstraction

This section introduces a method to systematically manage the combinatorial complexity that occurs when enumerating UCCAs. First, a simple example illustrates the need for the process. Next, a process of abstraction is developed to solve the problem.

[6] This defines discrete commands as different than continuous commands. As such, a discrete command cannot be treated as, or converted to, a continuous command in later steps of the hazard analysis. If a discrete command must be changed to a continuous command, the Type 4 UCA and Type 3-4 UCCA identification must be re-addressed with this change.

Combinatorial Complexity of UCCA Enumeration

Consider the following multi-UAS system concept used in the STPA example in Chapter 2.4 [143]. An operator controls two collaborative UAS by specifying mission tasks for the collective team to perform. These tasks represent control actions the UAS must provide to the shared mission process, such as *jamming* a radar and *striking* a target. In this example, the system enters a hazardous state if the team strikes the target without jamming the radar or if multiple UAS jam the radar simultaneously.

Figure 4-10 shows the control structure for this simple example. The two UAS can provide both the *jam* and the *strike* commands. They coordinate with each other to determine which UAS will perform each task. The goal in this work is to systematically explore how combinations of UAS_1 and UAS_2 issuing the *jam* and *strike* commands may be unsafe.

Figure 4-10 also includes a generalized form of the model, with $n = 2$ controllers that can each issue the $m = 2$ different control actions. This representation is referenced in the algorithmic enumeration discussion below.

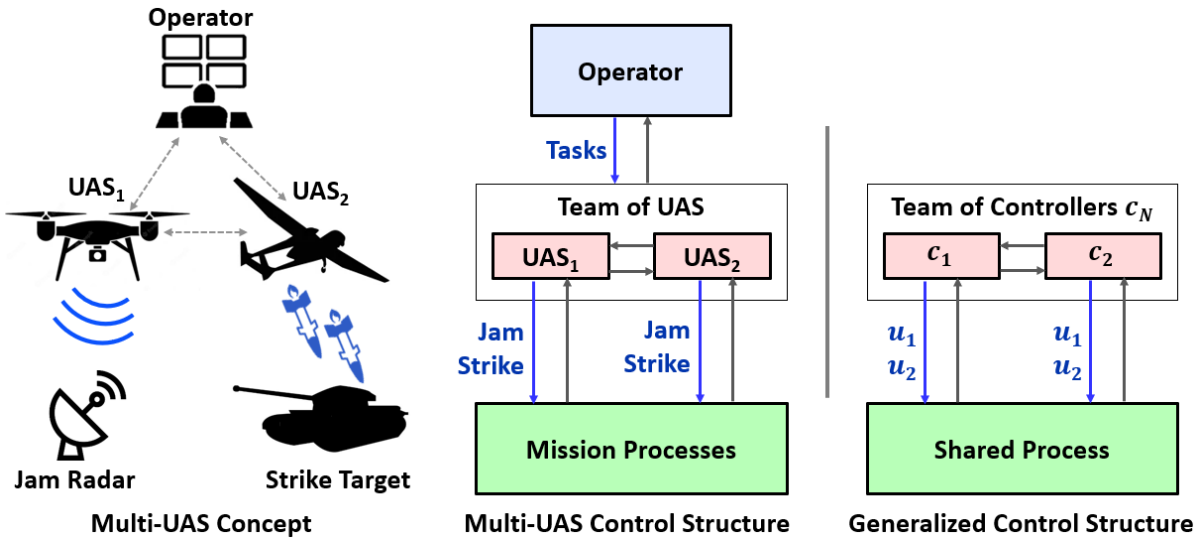

Figure 4-10. Illustrative Multi-UAS Team Example (left) and its Generalized Form (right)

The Types of UCCAs defined in the previous section help enumerate all the combinations of control actions provided by the two controllers in this example. Using the extended UCA structure, Type 1-2 UCCAs are formulated as:

1. c_1 **does not provide** $\{u_1$ *or* $u_2\}$; c_2 **does not provide** $\{u_1$ *or* $u_2\}$ when… *[H]*
2. c_1 **does not provide** $\{u_1$ *or* $u_2\}$; c_2 **does not provide** u_1 and **provides** u_2 when… *[H]*
3. c_1 **does not provide** $\{u_1$ *or* $u_2\}$; c_2 **provides** u_1 and **does not provide** u_2 when… *[H]*
4. …

It is simpler to enumerate combinations using a format inspired by truth tables, as shown in Table 4-1. The top row designates the controllers that issue the control actions in each subsequent row. Here, $\neg u$ means "does not provide u". Rows 1-3 in the table match one-to-one the three UCCAs specified in English above.

In this relatively simple problem, there are already 16 potential Type 1-2 UCCAs for the analyst to evaluate. There are even more potential Type 3-4 UCCAs. As shown in Figure 4-9, there are 8 ways to order the start and end of two different control actions. This example features 6 possible pairs of control actions, so there are $6 \times 8 = 48$ potential Type 3-4 UCCAs involving two actions. But the problem is actually more complicated because there can be sequences of three or even all four control actions to consider. Evaluating every potential combination becomes quickly intractable.

Table 4-1. Full Enumeration of Type 1-2 UCCAs for the Multi-UAS Example

#	c_1		c_2	
1	$\neg u_1$	$\neg u_2$	$\neg u_1$	$\neg u_2$
2	$\neg u_1$	$\neg u_2$	$\neg u_1$	u_2
3	$\neg u_1$	$\neg u_2$	u_1	$\neg u_2$
4	$\neg u_1$	$\neg u_2$	u_1	u_2
5	$\neg u_1$	u_2	$\neg u_1$	$\neg u_2$
6	$\neg u_1$	u_2	$\neg u_1$	u_2
7	$\neg u_1$	u_2	u_1	$\neg u_2$
8	$\neg u_1$	u_2	u_1	u_2
9	u_1	$\neg u_2$	$\neg u_1$	$\neg u_2$
10	u_1	$\neg u_2$	$\neg u_1$	u_2
11	u_1	$\neg u_2$	u_1	$\neg u_2$
12	u_1	$\neg u_2$	u_1	u_2
13	u_1	u_2	$\neg u_1$	$\neg u_2$
14	u_1	u_2	$\neg u_1$	u_2
15	u_1	u_2	u_1	$\neg u_2$
16	u_1	u_2	u_1	u_2

#	UAS_1		UAS_2		Context
1	¬jam	¬strike	¬jam	¬strike	*when…*
2	¬jam	¬strike	¬jam	strike	*when…*
3	¬jam	¬strike	jam	¬strike	*when…*
4	¬jam	¬strike	jam	strike	*when…*
5	¬jam	strike	¬jam	¬strike	*when…*
6	¬jam	strike	¬jam	strike	*when…*
7	¬jam	strike	jam	¬strike	*when…*
8	¬jam	strike	jam	strike	*when…*
9	jam	¬strike	¬jam	¬strike	*when…*
10	jam	¬strike	¬jam	strike	*when…*
11	jam	¬strike	jam	strike	*when…*
12	jam	¬strike	jam	strike	*when…*
13	jam	strike	¬jam	¬strike	*when…*
14	jam	strike	¬jam	strike	*when…*
15	jam	strike	jam	¬strike	*when…*
16	jam	strike	jam	strike	*when…*

Figure 4-11 illustrates the general problem with a team of n controllers that can issue m types of control actions to a shared process. The controllers may overlap in their ability to provide the same type of command for any number of these control actions. For generality, the figure shows the system having full overlap for all control actions.

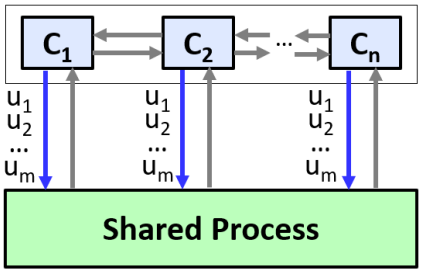

Figure 4-11. General Team of Multiple Controllers Issuing Multiple Control Actions

This team can provide up to p total control actions to the shared process, where p is defined by Equation (1). Here, $U_a(c_i) = 1$ if controller c_i can provide control action u_a, and $\boldsymbol{N}$ and $\boldsymbol{M}$ are the domains of all control actions and controllers respectively.

$$p = \sum U_i(c_a);\ \forall i\ \in \boldsymbol{N},\ \forall a \in \boldsymbol{M} \tag{1}$$

In the general problem, there are $d^{T12} = 2^p$ possible Type 1-2 UCCA combinations of n controllers *providing* or *not providing* each of the m control actions. This number grows exponentially with n and m. The number of Type 3-4 UCCA permutations, or ordered sequences, in which up to any of these p control actions can be *started* and *ended* relative to one another grows even faster, as found in Equation (2).

$$d^{T34} = \sum_{k=2}^{p} \frac{2^k p!}{(p-k)!} \tag{2}$$

Table 4-2 illustrates the total number of UCCAs for different hypothetical teams. For simplification, here every controller on the team can provide every control action, so $p = n \times m$.

Table 4-2. Number of UCCAs Enumerable for Different Hypothetical Teams

Controllers n	Types of Control Actions m	Total Control Actions p	Type 1-2 UCCAs d^{T12}	Type 3-4 UCCAs d^{T34}	Total UCCAs $d^{T12} + d^{T34}$
2	1	2	4	8	12
2	2	4	16	624	640
2	3	6	64	75,960	76,024
2	4	8	256	17,017,952	17,018,208
3	1	3	8	72	80
3	2	6	64	75,960	76,024
3	3	9	512	3.06×10^8	3.06×10^8
3	4	12	4096	3.23×10^{12}	3.23×10^{12}

As shown, teams of even modest sizes produce too many potential UCCAs to enumerate fully. It is not practical for a human analyst to identify the context(s) in which all these UCCAs are unsafe, and then develop causal scenarios to explain how they could occur. Such an effort would also be inefficient as it would produce similar information repeated across multiple similar UCCAs. Finally, the volume of data would be too overwhelming for designers to derive useful engineering and assurance decisions. Simplification is necessary.

Process of Abstraction to Manage Combinatorial Complexity

The ability to use abstraction to manage complexity is one of the key strengths of system-theoretic approaches. As shown below, collaborative systems can be systematically abstracted to reduce the combinatorial complexity associated with enumerating UCCAs. The approach, summarized in Figure 4-12, helps identify the context in which multiple control actions are unsafe together at a higher level of abstraction, where fewer combinations exist. Then, only those combinations found to be unsafe need to be further explored through refinement.

The approach begins with two different model representations, Abstractions 1a and 1b, which are shown one level up from the problem formulation in Figure 4-12. Each one focuses on a

different way to combine multiple control actions. The following discussion explains each of these abstractions and demonstrates how they enumerate UCCAs in the Multi-UAS example above. This is a necessary step to then abstract the model further to Abstraction 2a and 2b to handle more complex systems, as addressed later in Section 4.2.3.

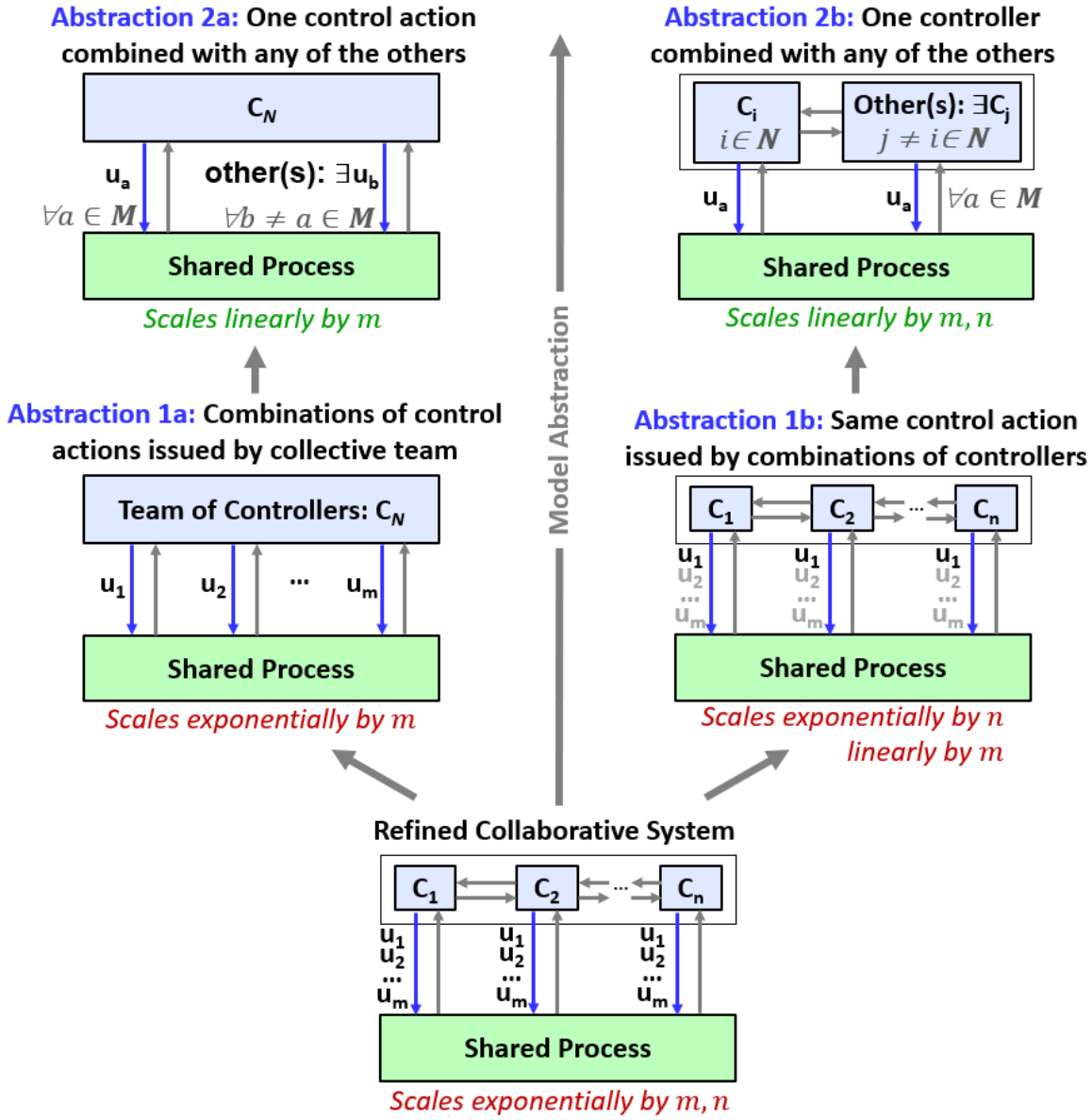

Figure 4-12. Managing Combinatorial Complexity Using Abstraction

UCCAs in Abstractions 1a: Combinations of Control Actions by Collective Team

Abstraction 1a combines the multiple controllers that share authority over a process into one collective controller (see Figure 4-12). This helps to identify the unsafe combinations of different types of control actions issued by the overall team to the process.

The use of this abstraction is predicated on several conditions. The system must exhibit shared authority between multiple controllers over a common process, or similarly, over different

interdependent subprocesses. It is also only useful if there are multiple different types of control actions to consider, or in other words for a set of control actions ($\{u_1, \ldots, u_m\} \mid m > 1$). However, it does not require the controllers to have any overlap in the types of control actions they provide.

For example, if in the multi-UAS system in Figure 4-10, only UAS_1 can *jam*, and only UAS_2 can *strike*, Abstraction 1a can be applied to explore combinations of these two commands issued by the collective team. Conversely, if the two UAS can only *jam*, and no UAS can *strike*, then this abstraction is not useful. Although in this case, Abstraction 1b is, as will be described later.

Table 4-3 shows the Type 1-2 UCCAs generated using Abstraction 1a for the multi-UAS example as defined in Figure 4-10. The table shows the combinations of control actions provided or not provided by the collective controller team, c_N.

Table 4-3. Abstraction 1a Type 1-2 UCCAs for Multi-UAS Example

#	c_N	
1	$\neg u_1$	$\neg u_2$
2	$\neg u_1$	u_2
3	u_1	$\neg u_2$
4	u_1	u_2

#	UAS Team		Context
1	¬jam	¬strike	*when…*
2	¬jam	strike	*when…*
3	jam	¬strike	*when…*
4	jam	strike	*when…*

Table 4-4 lists the Abstraction 1a Type 3-4 UCCAs, or sequences in which the collective team can start and end its control actions. In the table, $S(u)$ means "start control action u", $E(u)$ means "end u", and **F** is the Linear Temporal Logic operator for *Some **F**uture Step* [207]. The table has two columns headed by controllers, and it is read as the first controller starts/ends the control action in any row before the second controller listed starts/ends its control action in the same row. In this case, the controller is the collective team, c_N, so it is the same in both columns.

Table 4-4. Abstraction 1a Type 3-4 UCCAs for Multi-UAS Example

#	c_N	**F** c_N
1	$S(u_1)$	$S(u_2)$
2	$S(u_1)$	$E(u_2)$
3	$E(u_1)$	$S(u_2)$
4	$E(u_1)$	$E(u_2)$
5	$S(u_2)$	$S(u_1)$
6	$S(u_2)$	$E(u_1)$
7	$E(u_2)$	$S(u_1)$
8	$E(u_2)$	$E(u_1)$

#	Team	before Team	Context
1	*starts* jam	*starts* strike	*when…*
2	*starts* jam	*ends* strike	*when…*
3	*ends* jam	*starts* strike	*when…*
4	*ends* jam	*ends* strike	*when…*
5	*starts* strike	*starts* jam	*when…*
6	*starts* strike	*ends* jam	*when…*
7	*ends* strike	*starts* jam	*when…*
8	*ends* strike	*ends* jam	*when…*

These tables can be generated using automation. The human analyst then evaluates each item and determines if there are any context(s) in which that combination can lead to hazard(s). For instance, based on assumptions in the example, item 2 in Table 4-3 can be written as:

> **UCCA 1**: UAS Team **does not provide** jam and **provides** strike *when an enemy radar is surveilling the target area [H1]*.

Similarly, item 3 in Table 4-4 forms the following UCCA:

UCCA 2: UAS Team **ends providing** jam before it **starts providing** strike *when an enemy radar is surveilling the target area [H1]*.

In each case, the abstracted team can later be refined, again using automation, to explore how different combinations of controllers can issue control actions that lead to that collective UCCA output. The context and hazard traceability generated at the higher level are carried over to the refined UCCA. For example, UCCA 1 can be refined as:

UCCA 1.1: UAS_1 **does not provide** {jam or strike}; UAS_2 **does not provide** jam and **provides** strike *when an enemy radar is surveilling target area [H1]*.

UCCA 1.2: UAS_1 **does not provide** jam and **provides** strike; UAS_2 **does not provide** {jam or strike} *when an enemy radar is surveilling target area [H1]*.

UCCA 1.3: UAS_1 **does not provide** jam and **provides** strike; UAS_2 **does not provide** jam and **provides** strike *when an enemy radar is surveilling target area [H1]*.

The same process of refinement can be applied to UCCA 2. The reason for refining the UCCA is that the causal factors later analyzed by developing loss scenarios may be different for the different controllers involved. The refinement of UCCAs is further explained and formalized in Section 4.2.4.

UCCAs in Abstractions 1b: Combinations of Controllers Issuing Shared Control Action

Abstraction 1b helps to determine the unsafe combinations of controllers issuing a common control action. As reflected in Figure 4-12, it represents the multiple controllers and focuses only on one common type of control action at a time.

As was the case in Abstraction 1a, the application of 1b is also predicated on the *shared authority* of multiple controllers over a common process. There must also be some overlap between at least two of the controllers in their ability to issue a common type of control action. This overlap may be enabled by *dynamic authority* or *transfer of authority* over that control action, but that is not always the case. For example, multiple aircraft flying in formation have a standing shared control authority over aircraft separation, which can be analyzed using this abstraction.

The abstraction does not require all controllers to have common authority over all tasks. In the Multi-UAS example (Figure 4-10), if only UAS_1 can *strike*, but both UAS can *jam*, then Abstraction 1b can be applied to analyze the combination of multiple controllers issuing the *jam* command. Conversely, if only UAS_1 can *strike* and only UAS_2 can *jam*, then this abstraction is not relevant. Although in this case, it should still be analyzed with respect to Abstraction 1a as previously described.

Table 4-5 and Table 4-6 enumerate Type 1-2 and Type 3-4 combinations of controllers issuing the same control action using Abstraction 1b. The tables focus on control action u_1, the *jam* command in the multi-UAS example, and would be duplicated for u_2, *strike*. Here, $U(x)$ means "control action u is provided by controller x", where x is enumerated as c_1 and c_2. All other conventions are consistent with those previously defined.

Table 4-5. Abstraction 1b Type 1-2 UCCAs for Multi-UAS Example

#	$U_1(x)$		Common Issues
1	$\neg U_1(c_1)$	$\neg U_1(c_2)$	control gap
2	$\neg U_1(c_1)$	$U_1(c_2)$	controller-task mismatch
3	$U_1(c_1)$	$\neg U_1(c_2)$	
4	$U_1(c_1)$	$U_1(c_2)$	control overlap

#	jam provided by		Context
1	¬UAS$_1$	¬UAS$_2$	*when…*
2	¬UAS$_1$	UAS$_2$	*when…*
3	UAS$_1$	¬UAS$_2$	*when…*
4	UAS$_1$	UAS$_2$	*when…*

Table 4-6. Abstraction 1b Type 3-4 UCCAs for Multi-UAS Example

#	$U_1(x)$	**F** $U_1(x)$	Common Issues
1	$S(U_1(c_1))$	$S(U_1(c_2))$	
2	$S(U_1(c_1))$	$E(U_1(c_2))$	handoff overlap
3	$E(U_1(c_1))$	$S(U_1(c_2))$	handoff gap
4	$E(U_1(c_1))$	$E(U_1(c_2))$	
5	$S(U_1(c_2))$	$S(U_1(c_1))$	
6	$S(U_1(c_2))$	$E(U_1(c_1))$	handoff overlap
7	$E(U_1(c_2))$	$S(U_1(c_1))$	handoff gap
8	$E(U_1(c_2))$	$E(U_1(c_1))$	

#	jam by	before jam by	Context
1	UAS$_1$ *starts*	UAS$_2$ *starts*	*when…*
2	UAS$_1$ *starts*	UAS$_2$ *ends*	*when…*
3	UAS$_1$ *ends*	UAS$_2$ *starts*	*when…*
4	UAS$_1$ *ends*	UAS$_2$ *ends*	*when…*
5	UAS$_2$ *starts*	UAS$_1$ *starts*	*when…*
6	UAS$_2$ *starts*	UAS$_1$ *ends*	*when…*
7	UAS$_2$ *ends*	UAS$_1$ *starts*	*when…*
8	UAS$_2$ *ends*	UAS$_1$ *ends*	*when…*

The systematic consideration of how different controllers provide, don't provide, start, and end a particular command highlights potential gaps, overlaps, mismatches, and handoff issues that are common in shared control. For example, item 1 in Table 4-5, which involves no controller providing the command, may be due to a gap in task allocation associated with *dynamic authority*. Items 2 and 3 explore how the "wrong" controller provides an action as can occur in a controller-task mismatch. Item 4 considers multiple controllers providing the action, which may lead to control conflicts.

Table 4-6 highlights related issues. In items 2 and 6, one controller starts providing the command before another controller ends it. This may occur in a *transfer of authority* when the handoff involves an excessive period of overlap between the two controllers, which can result in conflicting control. Items 3 and 7 reflect the opposite problem when one controller ends before the next starts. This can lead to a period of gap in authority when the process is not controlled. The overlap and gap in control are visible in items 2 and 3, respectively, in Figure 4-9.

As mentioned for Abstraction 1a, the tables can be generated using automation, and an analyst can then specify if there are any context(s) and hazard(s) that apply to each item. For example, item 4 in Table 4-5 can generate the following UCCA. In this case, there is no need to refine the UCCA as it already provides the controllers and control actions involved.

> **UCCA 3**: UAS$_1$ **provides** jam and UAS$_2$ **provides** jam *when both controllers interfere with each other [H2]*.

Summary of Abstractions 1a and 1b

By applying Abstractions 1a and 1b to the multi-UAS example, the number of potential UCCAs is reduced from 640 in the fully enumerated set, as listed in Table 4-2 for $n = 2$ and $m = 2$, down to 36. There are now 12 Type 1-2 UCCAs and 24 Type 3-4 UCCAs for a human to analyze.

Unfortunately, while the number of UCCAs grows more slowly than in the fully refined problem, at this level of abstraction, it still scales exponentially. Abstraction 1a is, by definition, combinatorial in the number of different types of control actions m. Abstraction 1b grows exponentially with the number of controllers n. And in both cases, the permutations of Type 3-4 UCCAs scale even faster once they involve more than the simple pairwise comparisons illustrated so far. Further simplification is necessary.

4.2.3 Linearizing Growth by Abstracting Further

The method of abstraction applied in the previous section provides the first step to mitigate exponential scaling in the number of UCCAs. This section introduces an additional level of abstraction, shown at the top of Figure 4-12, which linearizes the growth of potential UCCAs for collaborative systems of any number of controllers and control actions.

The previous multi-UAS example is slightly modified to illustrate the process. Consider now a team of three unmanned systems (UxS), consisting of 2 UAS and 1 ground robot ($n = 3$). Each controller can execute the same types of control actions: *jam*, *strike*, and *track* a target ($m = 3$). Figure 4-13 shows the updated control structure for this concept alongside its generalized form.

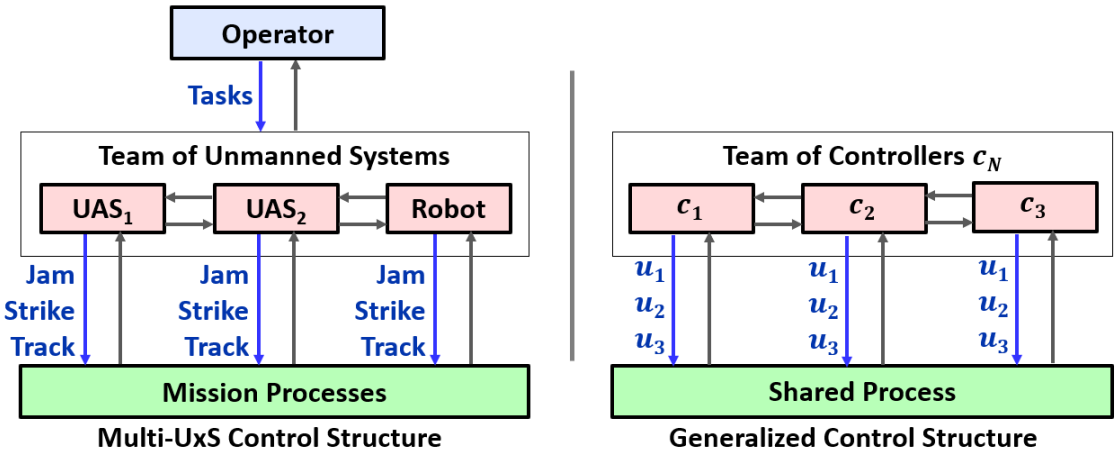

Figure 4-13. Updated Multi-UxS Control Structure (left) and its Generalized Form (right)

To reduce the number of potential UCCAs, the system is first represented using Abstractions 1a and 1b as described in the previous section. Each model is then further abstracted, respectively, to Abstractions 2a and 2b, as shown in Figure 4-12.

Figure 4-14 shows this process for the multi-UxS example. The figure only depicts one iteration of Abstractions 2a and 2b centered on control action u_1. It also explains how to reiterate the cases for u_2 and u_3. Finally, the figure clarifies that a system can be directly represented by Abstractions 2a and 2b. Abstractions 1a and 1b simply provide the system-theoretic foundation for the higher-level representation, but no analysis is necessary at that level.

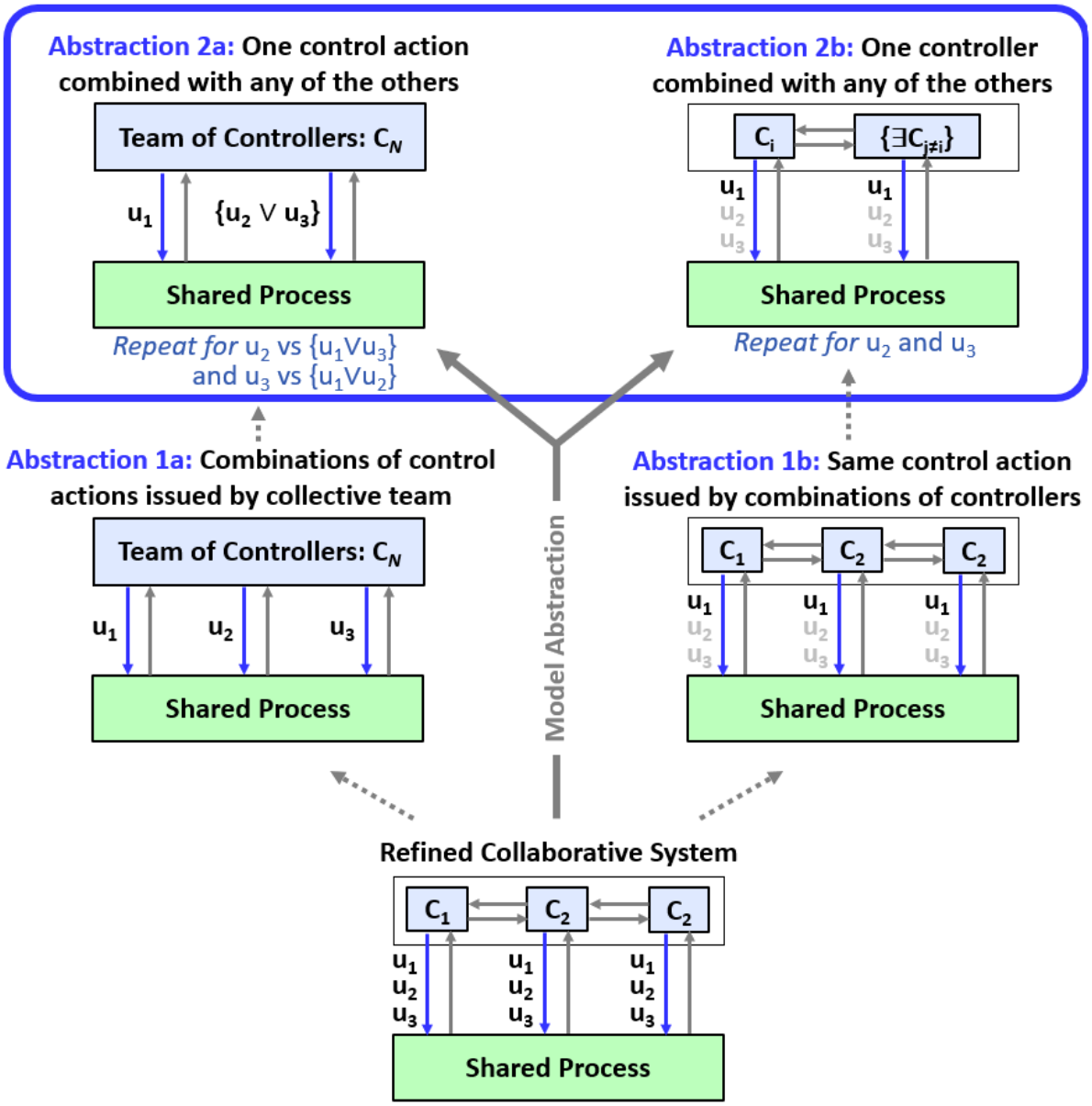

Figure 4-14. Abstraction of a Collaborative System to Models 2a and 2b

UCCAs in Abstraction 2a

Abstraction 2a is based on Abstraction 1a, and it iteratively considers any one of the control actions in combination with any of the others abstracted together (see Figure 4-14). At this level, Type 1-2 UCCAs explore how the collective team provides (or does not provide) any one control action and provides (or does not provide) any of the others. One iteration of this process comparing u_1 with $\{u_2 \vee u_3\}$ for the modified multi-UxS example is:

1. c_N **does not provide** u_1 and **does not provide** $\{u_2$ or $u_3\}$ when… *[H]*
2. c_N **does not provide** u_1 and **provides** $\{u_2$ or $u_3\}$ when… *[H]*
3. c_N **provides** u_1 and **does not provide** $\{u_2$ or $u_3\}$ when… *[H]*
4. c_N **provides** u_1 and **provides** $\{u_2$ or $u_3\}$ when… *[H]*

In these statements, $\{u_2$ or $u_3\}$ is true under the following conditions: u_2 alone is true, u_3 alone is true, or u_2 and u_3 are both true. Two additional iterations compare u_2 with $\{u_1 \vee u_3\}$ and u_3 with $\{u_1 \vee u_2\}$. Their formulations are comparable to those listed above. The full set of potential Type 1-2 UCCAs for Abstraction 2a of the multi-UxS system is shown in Table 4-7.

Table 4-7. Abstraction 2a Type 1-2 UCCAs for Multi-UxS Example

#	c_N	
1	$\neg u_1$	$\neg\{u_2 \vee u_3\}$
2	$\neg u_1$	$\{u_2 \vee u_3\}$
3	u_1	$\neg\{u_2 \vee u_3\}$
4	u_1	$\{u_2 \vee u_3\}$
5	$\neg u_2$	$\neg\{u_1 \vee u_3\}$
6	$\neg u_2$	$\{u_1 \vee u_3\}$
7	u_2	$\neg\{u_1 \vee u_3\}$
8	u_2	$\{u_1 \vee u_3\}$
9	$\neg u_3$	$\neg\{u_1 \vee u_2\}$
10	$\neg u_3$	$\{u_1 \vee u_2\}$
11	u_3	$\neg\{u_1 \vee u_2\}$
12	u_3	$\{u_1 \vee u_2\}$

#	UxS Team		Context
1	¬jam	¬any in{strike, track}	*when…*
2	¬jam	any in{strike, track}	*when…*
3	jam	¬any in{strike, track}	*when…*
4	jam	any in{strike, track}	*when…*
5	¬strike	¬any in{jam, track}	*when…*
6	¬strike	any in{jam, track}	*when…*
7	strike	¬any in{jam, track}	*when…*
8	strike	any in{jam, track}	*when…*
9	¬track	¬any in{jam, strike}	*when…*
10	¬track	any in{jam, strike}	*when…*
11	track	¬any in{jam, strike}	*when…*
12	track	any in{jam, strike}	*when…*

Type 3-4 UCCAs follow a similar concept. They represent ways in which the collective team may start or end any one control action before starting or ending any of the others. One of three iterations of Type 3-4 UCCAs in Abstraction 2a is listed below and the full set for the multi-UxS example is provided in Table 4-8.

1. c_N **starts providing** u_1 before it **starts providing** $\{u_2$ or $u_3\}$ when… *[H]*
2. c_N **starts providing** u_1 before it **ends providing** $\{u_2$ or $u_3\}$ when… *[H]*
3. c_N **ends providing** u_1 before it **starts providing** $\{u_2$ or $u_3\}$ when… *[H]*
4. c_N **ends providing** u_1 before it **ends providing** $\{u_2$ or $u_3\}$ when… *[H]*

In the statements above, if either u_2 or u_3 is a discrete command, it must be removed from consideration in specifying the UCCA contexts that involve "ends providing $\{u_2$ or $u_3\}$". This removal can be automated. For example, if u_2 is a discrete command and u_3 is a continuous command, then the second line must be interpreted as:

2. c_N **starts providing** u_1 before it **ends providing** u_3 when… *[H]*

Similarly, if both u_2 and u_3 are discrete commands, the statement is removed altogether. However, statements 1 and 3, which involve starting these discrete commands, would still be valid to consider.

Table 4-8. Abstraction 2a Type 3-4 UCCAs for Multi-UxS Example

#	c_N	**F** c_N
1	$S(u_1)$	$S(\{u_2 \vee u_3\})$
2	$S(u_1)$	$E(\{u_2 \vee u_3\})$
3	$E(u_1)$	$S(\{u_2 \vee u_3\})$
4	$E(u_1)$	$E(\{u_2 \vee u_3\})$
5	$S(u_2)$	$S(\{u_1 \vee u_3\})$
6	$S(u_2)$	$E(\{u_1 \vee u_3\})$
7	$E(u_2)$	$S(\{u_1 \vee u_3\})$
8	$E(u_2)$	$E(\{u_1 \vee u_3\})$
9	$S(u_3)$	$S(\{u_1 \vee u_2\})$
10	$S(u_3)$	$E(\{u_1 \vee u_2\})$
11	$E(u_3)$	$S(\{u_1 \vee u_2\})$
12	$E(u_3)$	$E(\{u_1 \vee u_2\})$
13*	$S(\{u_2 \vee u_3\})$	$S(u_1)$
…	…	…
24*	$E(\{u_1 \vee u_2\})$	$E(u_3)$

#	Team	before Team	Context
1	*starts* jam	*starts* any in{strike, track}	*when…*
2	*starts* jam	*ends* any in{strike, track}	*when…*
3	*ends* jam	*starts* any in{strike, track}	*when…*
4	*ends* jam	*ends* any in{strike, track}	*when…*
5	*starts* strike	*starts* any in{jam, track}	*when…*
6	*starts* strike	*ends* any in{jam, track}	*when…*
7	*ends* strike	*starts* any in{jam, track}	*when…*
8	*ends* strike	*ends* any in{jam, track}	*when…*
9	*starts* track	*starts* any in{jam, strike}	*when…*
10	*starts* track	*ends* any in{jam, strike}	*when…*
11	*ends* track	*starts* any in{jam, strike}	*when…*
12	*ends* track	*ends* any in{jam, strike}	*when…*
13*	*starts* any in{strike, track}	*starts* jam	*when…*
…	…	…	…
24*	*ends* any in{jam, strike}	*ends* track	*when…*

**Items 13-24 are the reverse sequences of Items 1-12 respectively*

As in Abstractions 1a and 1b, these tables can also be generated by automation, and the same approach is employed to identify and refine UCCAs (see Section 4.2.2). However, one notable difference is that in Abstraction 2a the UCCA must also specify what control actions in the abstracted set are relevant to the context. This determination must be made by the human analyst because the context is defined by the analyst. For example, item 10 in Table 4-7 may produce:

> **UCCA 4**: UAS Team **does not provide** track and **provides** strike *when an enemy must be tracked as a strike occurs to ensure custody of the target [H3]*.

This UCCA is uncovered by analyzing the ¬*track* and {*jam* ∨ *strike*} combination of commands. However, in the set {*jam* ∨ *strike*}, only the *strike* command is relevant. Under the context specified, it does not matter if *jam* is provided or not. By designating the relevant commands, the refined UCCA only includes the different controller options that lead to collectively not tracking and striking (see Table 4-9). The exclusion of *jam* eliminates additional combinations that are invariant to the context and subsequent causal analysis of the UCCA.

Table 4-9. Refinement of Example UCCA 4

UCCA	Team	Team	Context
4	¬track	{~~jam~~ ∨ strike}	*when enemy must be tracked as strike occurs [H3]*

Refined	UAS_1	UAS_2	robot	Same Context and Hazard
4.1	¬track	¬track	strike	*when enemy must be tracked as strike occurs [H3]*
4.2	¬track	strike	¬track	*when enemy must be tracked as strike occurs [H3]*
4.3	strike	¬track	¬track	*when enemy must be tracked as strike occurs [H3]*
4.4	¬track	strike	strike	*when enemy must be tracked as strike occurs [H3]*
4.5	strike	¬track	strike	*when enemy must be tracked as strike occurs [H3]*
4.6	strike	strike	¬track	*when enemy must be tracked as strike occurs [H3]*
4.7	strike	strike	strike	*when enemy must be tracked as strike occurs [H3]*

In other UCCAs, the analyst may choose to include all of the control actions listed in the combination. For example, a version of the previous UCCA 1 updated for this modified UxS example may form UCCA 5 below.

> **UCCA 5**: UAS Team **does not provide** jam and **provides** track and strike *when an enemy radar is surveilling the target area [H1]*.

UCCAs in Abstraction 2b

Abstraction 2b follows a similar logic as in 2a and uses Abstraction 1b as a foundation to consider how any one controller in combination with any of the others abstracted together may issue a common control action (see Figure 4-14). Here, Type 1-2 UCCAs explore ways the one controller and the others provide or do not provide the action, as shown in Table 4-10. The process is reiterated for each control action.

Table 4-10. Abstraction 2b Type 1-2 UCCAs for Multi-UxS Example

#	$U_1(x)$		Common Issues
1	$\neg U_1(c_i)$	$\neg U_1(\{c_{j1} \vee c_{j2}\})$	gap
*	$\neg U_1(c_i)$	$U_1(\{c_{j1} \vee c_{j2}\})$	
2	$U_1(c_i)$	$\neg U_1(\{c_{j1} \vee c_{j2}\})$	mismatch
3	$U_1(c_i)$	$U_1(\{c_{j1} \vee c_{j2}\})$	overlap

#	jam provided by		Context
1	¬any one	¬any of others	*when…*
*	¬any one	any of others	*when…*
2	any one	¬any of others	*when…*
3	any one	any of others	*when…*

**Does not need to be considered, included in items 2 and 3 (see discussion)*

A key difference in how UCCAs are defined in Abstraction 2b, in this work, is that the specific controllers do not need to be listed in the abstracted form. For the systems explored in this research, it is assumed that the context in which a control action, or a combination of control actions, is unsafe is controller agnostic. As such, the controllers listed in the abstracted UCCA (e.g., those in Table 4-10) are generalized, which reduces the number of combinations that need to be analyzed at this level.

However, it is important to note that the specific controllers must be considered in the refinement of the UCCA. This is a necessary step to explore how those different controllers can contribute to that higher-level output. By listing these options, the causal factors that are controller-specific can later be identified in the last step of the hazard analysis, scenario development. These factors would otherwise be missed. The process of refining a UCCA can be fully automated, and the context and hazard identified by the human analyst at the higher level are carried over to the refined level, as shown below.

An example of a UCCA found in the multi-UxS system using Abstraction 2b follows. The last combination in Table 4-10, in which multiple controllers provided the *jam* command, is refined in Table 4-11.

> **UCCA 6**: any one C_i **provides** jam and any other C_j **provide(s)** jam *when multiple controllers interfere with each other [H2]*.

Table 4-11. Refinement of Example UCCA 5

UCCA	C_i	Any other C_j		Context
6	jam	jam		*when multiple controllers mutually interfere [H2]*
Refined	UAS_1	UAS_2	robot	Same Context and Hazard
6.1		jam	jam	*when multiple controllers mutually interfere [H2]*
6.2	jam		jam	*when multiple controllers mutually interfere [H2]*
6.3	jam	jam		*when multiple controllers mutually interfere [H2]*
6.4	jam	jam	jam	*when multiple controllers mutually interfere [H2]*

The refinement in Table 4-11 assumes all controllers can issue all control actions. If instead, only UAS_1 and UAS_2 could provide the *jam* command, and not the robot, then item 6.3 would be the only option in the refinement.

Another advantage to generalizing the controllers in the UCCA identification is that the second row labeled (*) in Table 4-10 can be skipped. This line represents one controller c_i not providing the command when any of the others in $\{c_{j\neq i}\}$ provide it. If only one controller in $\{c_{j\neq i}\}$ provides the command, that is addressed by item 2 of the table. If multiple controllers in $\{c_{j\neq i}\}$ provide it, that is considered in item 3.

Despite the benefits, there may be circumstances in which an engineering team chooses not to make the assumption that context is controller agnostic. In such cases, the same mechanism used to enumerate multiple different control actions in Abstraction 2a can be applied to enumerate multiple specific controllers. All four items per iteration in Table 4-10 would need to be considered, the process would be reiterated to compare each controller to the others, and the analyst could specify which controllers are relevant to the contexts identified. The reasons for selecting this approach are beyond the scope of this dissertation.

Type 3-4 UCCAs in this abstraction explore the sequences in which any one controller and any other start or end a common control action. Table 4-12 shows the process for one control action (u_1), and similar iterations are necessary for each of the other control actions.

By again assuming that the context is controller agnostic, the process only needs to consider any one of the other controllers in $\{c_{j\neq i}\}$ at a time. Similar to above, if an engineering team does not want to make this assumption, it can enumerate options such as c_i starts/ends u_1 before any controllers in $\{c_{j1}, \ldots, c_{j(n-1)}\}$ start/end u_1. The enumeration would also need to include the reverse sequences, and the analyst would specify the controllers relevant to the context identified.

Table 4-12. Abstraction 2b Type 3-4 UCCAs for Multi-UxS Example

#	$U_1(x)$	**F** $U_1(x)$	Common Issues
1	$S(U_1(c_i))$	$S(U_1(c_j))$	
2	$S(U_1(c_i))$	$E(U_1(c_j))$	handoff overlap
3	$E(U_1(c_i))$	$S(U_1(c_j))$	handoff gap
4	$E(U_1(c_i))$	$E(U_1(c_j))$	

#	jam by	before jam by	Context
1	any one *starts*	any other *starts*	*when…*
2	any one *starts*	any other *ends*	*when…*
3	any one *ends*	any other *starts*	*when…*
4	any one *ends*	any other *ends*	*when…*

As discussed for Abstraction 1b, Abstraction 2b also highlights the same common issues with shared control that may contribute to entering a hazardous state. The ability to systematically identify potential control gaps, overlaps, mismatches, and unsafe handoffs as listed in Table 4-10 and Table 4-12 is a key strength of the overall UCCA approach.

Assumptions and Limitations

The abstractions employed to enumerate UCCAs presented in this section provide tractability to an otherwise combinatorial problem. However, the method involves certain assumptions and has some limitations that are important to understand.

The following are the key assumptions. First, the abstractions are predicated on multiple controllers sharing authority over a common process or over different interdependent subprocesses. Second, Abstractions 1a and 2a assume these controllers, collectively, can provide multiple types of control actions to the shared process. Third, Abstractions 1b and 2b assume multiple controllers can provide the same type of control action to the process. Fourth, in this work, the context of the UCCA is assumed to be agnostic to the controllers that provide the actions. Mechanisms to relax this assumption are addressed in the discussion above. Finally, it is assumed a human analyst is able to identify the unsafe context of a UCCA given the abstracted combination of control actions or controllers.

The main objective of the abstractions is to reduce the number of potential UCCAs to consider in arbitrarily complex systems. However, in the multi-UxS example (Figure 4-13), the process generates more Type 1-2 UCCAs (provide / not provide) in Abstraction 2a than in Abstraction 1a. In Abstraction 2a, the UCCA count grows linearly by $4m$, instead of exponentially by 2^m in 1a, where m is the number of different types of control actions (here, $m = 3$).

The numerical advantage of the additional abstraction is realized when $m > 4$. This means an argument can be made to enumerate Type 1-2 UCCAs using Abstraction 1a instead of 2a for systems with 4 or fewer control actions. Anecdotally, the author found it easier cognitively to explore combinations of more than 3 control actions using Abstraction 2a. Similarly, because controllers are defined generally, Abstraction 2b is advantageous over Abstraction 1b when there are three or more controllers.

Abstractions 2a and 2b provide even greater benefits for Type 3-4 UCCAs (start/end before others start/end). The abstraction leverages some of the numerical advantages of pairwise analysis by avoiding permutations of sequences greater than two. However, by maintaining consideration for all controllers or control actions at a time, the abstraction retains a broader analytical scope than if the problem was reduced to only evaluate pairs. Such a reduction could never consider interactions that involve more than two controllers or control actions.

One limitation is that some control combinations may be missed due to the method of abstraction. Table 4-13 provides representative examples of UCCAs that are omitted from the abstracted enumeration and an explanation for why they are not covered.

In all cases, simpler combinations that address a subset of the overall interaction are identified. However, as informed by Systems Theory, the aggregate of these reduced cases may not necessarily represent the behavior of the full interaction. Still, these cases may help analysts consider the more complex combinations that are missed by the abstraction.

Table 4-13. Examples of Control Combinations Not Addressed Using Full Abstraction

Missed UCCA	Reason it is Missed
C_N **provides** u_1 and u_2 and **does not provide** u_3 and u_4 $u_1 \wedge u_2 \wedge \neg u_3 \wedge \neg u_4$	Abstraction 2a considers whether 1 control action is provided (or not), and if any of the others are provided (or not). However, this UCCA is considered in Abstraction 1a. Simpler combinations addressed in Abstraction 2a: $(u_1 \wedge u_2 \wedge \neg u_3)$; $(u_1 \wedge u_2 \wedge \neg u_4)$; $(u_1 \wedge \neg u_3 \wedge \neg u_4)$; $(u_2 \wedge \neg u_3 \wedge \neg u_4)$
C_i **ends** u_1 before (C_j **starts** u_2 and C_j **ends** u_3)	Similar to above, the abstraction applies Start or End to the other commands together. Simpler combinations addressed: C_i **ends** u_1 before C_j **starts** u_2; C_i **ends** u_1 before C_j **ends** u_3
C_i **ends** u_1 before (C_j **starts** u_1 and C_i **starts** u_2)	The combination bridges over the initial decision to look at combinations of different control actions, and combinations of controllers issuing the same control action. Simpler combinations addressed: C_i **ends** u_1 before C_j **starts** u_1; C_i **ends** u_1 before C_i **starts** u_2
C_i **ends** u_1 before C_j **starts** u_2 before C_i **starts** u_3	This is a sequence of three events, which is beyond what is considered. Simpler combinations addressed: C_i **ends** u_1 before {C_j **starts** u_2 and C_i **starts** u_3}**;** C_j **starts** u_2 before C_i **starts** u_3
C_1 **ends** u_1 before (C_2 **starts** u_1 and C_3 **starts** u_1)	Abstraction 2b Type 3-4 UCCAs only consider any one controller and any other one by assuming the context is controller agnostic (by choice only, see discussion) Simpler combinations addressed: C_1 **ends** u_1 before C_2 **starts** u_1; C_1 **ends** u_1 before C_3 **starts** u_1

To illustrate one of these missed combinations and its simpler cases considered, the second line of the table may represent the following ordered sequence for the multi-UxS example in Figure 4-13.

Missed: Any C_i **ends** jam before any others in C_j **start** strike and **ends** track *when …*

Simpler considered: Any C_i **ends** jam before any other C_j **starts** strike *when …*

Simpler considered: Any C_i **ends** jam before any other C_j **ends** track *when …*

The last item in Table 4-13 occurs because of the decision to assume the context of a UCCA is controller agnostic. Here, the missed UCCA does not need to be considered as long as the simpler combinations listed are addressed. However, as discussed in the previous subsection, the analysis team can choose not to make this assumption and recover the ability to find the missed UCCA. As such, this case represents a deliberate implementation decision rather than a limitation of the method.

Some of the other cases in the table may be found by arbitrarily extending the scope of the abstraction. For example, Type 1-2 UCCAs could have an additional step to compare any two controllers with all the others. Similarly, Type 3-4 UCCAs could consider some sequences of three changes in actions. These strategies may help if an analyst finds a pattern of combinations to consider that is just beyond the reach of the current approach.

However, if all higher-order combinations must be found, the guaranteed method to find them is to perform a full enumeration as described at the beginning of this section. Despite the limitations above, the overall approach to identifying UCCAs is shown in the Chapter 5 case study to be practical and find causal information that was previously not identified.

4.2.4 UCCA Identification Algorithm

The concepts developed in the previous subsections are now integrated into an end-to-end algorithm (Algorithm 1) to identify Unsafe Combinations of Control Actions (UCCAs). The algorithm introduces new concepts to reduce and prioritize the output set of UCCAs and improve the efficiency of scenario development.

Algorithm 1: UCCA Identification

Input: $\mathcal{A}$, $\mathcal{C}^{int}$, $\mathcal{S}$
Output: $\mathcal{U}^{abs}$, $\mathcal{U}^{ref}$ sets of UCCAs, abstracted & refined (Tuple)
// $\mathcal{A}$: what controller can provide what control action (Tuple)
// $\mathcal{C}^{int}$: set of interchangeable controllers (Set)
// $\mathcal{S}$: special interactions to consider in refinement (Tuple)

1. $\mathcal{C}^{abs} \leftarrow$ **Enumerate-Combinations** ($\mathcal{A}$) // [Table 4-14]*
2. for $x \in \mathcal{C}^{abs}$**
3. if Context($\mathcal{C}_x^{abs}$) $\neq \emptyset$**
4. $\mathcal{U}^{abs} = \mathcal{U}^{abs} \cup (\mathcal{C}_x^{abs}, context_x, \mathcal{U}_x^{rel})$**
5. for $x \in \mathcal{U}^{abs}$*
6. $\mathcal{C}_x^{ref} \leftarrow$ **Refine-Combinations** ($\mathcal{U}_x^{abs}$, $\mathcal{A}$, $\mathcal{S}$) // [Equation (17)]*
7. $\mathcal{C}_x^{ref\prime} \leftarrow$ **Prune-Equivalent**($\mathcal{C}_x^{ref}$, $\mathcal{C}^{int}$) // [Equation (21)]*
8. $\mathcal{U}_x^{ref} \leftarrow$ **Prioritize**($\mathcal{C}_x^{ref\prime}$, $\mathcal{S}$) // [Heuristic, see discussion]*
9. **return** ($\mathcal{U}^{abs}$, $\mathcal{U}^{ref}$)*

// * Step automated; ** Step performed by human

The UCCA Identification Algorithm includes portions that are automated (denoted by *) and others that are performed by a human analyst (**). A description of each component in the algorithm follows and includes reasoning for allocating that part of the process to the automation or the human. Some of the components are illustrated using the multi-UxS system example from Figure 4-13. A prototype tool that executes the automated portions of Algorithm 1 was developed in MATLAB and is employed in the case study in Chapter 5. The implementation of the tool is discussed where relevant.

Inputs

The first input to the algorithm is the authority tuple, $\boldsymbol{\mathcal{A}} = f(\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{A})$, which describes the set of control actions each controller can provide to the shared process. $\boldsymbol{\mathcal{A}}$ is derived from the control structure, and its components are:

> $\boldsymbol{N} \coloneqq$ vector of controllers that share authority over the process
>
> $\boldsymbol{M} \coloneqq$ vector of control actions that can be provided to the process
>
> $\boldsymbol{A} \coloneqq$ vector $\{A_a(c_i)\}$, $\forall a \in \boldsymbol{M}$, $\forall i \in \boldsymbol{N}$, for which elements are true if controller c_i can provide command u_a

The second input is the set of interchangeable controllers, $\boldsymbol{C^{int}}$. It is used to prune refined UCCAs that are considered equivalent to others in terms of potential causal factors. This topic is further addressed in the discussion about Line 7.

The third input is a set of special interactions, $\boldsymbol{S}$, encoded to add or remove refined UCCAs when enumerated. Special interactions can also be used to influence prioritization. They are further discussed in the descriptions of Lines 6 and 8.

Example: for the multi-UxS system in Figure 4-13, these terms are captured in Equations (3)-(7) below. $\boldsymbol{S}$ is addressed later in the discussion.

$$\boldsymbol{N} = \{UAS_1, UAS_1, robot\} \tag{3}$$

$$\boldsymbol{M} = \{jam, strike, track\} \tag{4}$$

$$\boldsymbol{A} = \{A_{jam}(UAS_1), A_{strike}(UAS_1), \dots, A_{track}(robot)\} = \{1,1,\dots,1\} \tag{5}$$

$$\boldsymbol{\mathcal{A}} = \begin{pmatrix} UAS_1 & jam & 1 \\ UAS_1 & strike & 1 \\ UAS_1 & track & 1 \\ UAS_2 & jam & 1 \\ UAS_2 & strike & 1 \\ UAS_2 & track & 1 \\ robot & jam & 1 \\ robot & strike & 1 \\ robot & track & 1 \end{pmatrix} \tag{6}$$

$$\boldsymbol{C^{int}} = \{UAS_1, UAS_2\} \tag{7}$$

Line 1: Automated Enumeration of Control Action Combinations

The first line in Algorithm 1 is a function that enumerates all the combinations of control actions by implementing the procedures introduced in Sections 4.2.1-4.2.3. It finds the Type 1-2 (provide/not provide) and the Type 3-4 (start/end before start/end) UCCAs using Abstractions 2a (combination of actions issued by the team) and 2b (combinations of controllers issuing a common action).

Table 4-14 formalizes this process. The first row outputs potential Type 1-2 UCCAs using Abstraction 2a in a format consistent with Table 4-7 for the multi-UxS example. These

combinations are provided by the collective team c_N of $\boldsymbol{N}$ controllers. The four cases are reiterated for every u_a in $\boldsymbol{M}$. For example, item (c) in that row describes the team providing u_a and not providing any of the other control actions, denoted by $\neg\exists u_b$.

The second row finds Type 1-2 UCCAs using Abstraction 2b like those shown in Table 4-8. The output keeps the controllers c_i and $\exists c_j$ general, where $\exists c_j$ denotes any the other controllers that are not c_i, and iterates for every u_a in $\boldsymbol{M}$. Here, item (c) in that row describes any one controller c_i providing u_a and any of the other controllers, $\exists c_j$, providing the same action.

Table 4-14. Formalized Method of Enumerating Combinations of Control Actions

Abstraction & UCCA Type	Cases to Consider		Enumerate Each Case
Abs 2a Type 1-2	a. $\neg u_a \wedge \neg\exists u_b$ b. $\neg u_a \wedge \exists u_b$ c. $u_a \wedge \neg\exists u_b$ d. $u_a \wedge \exists u_b$	given $U_a(c_N), U_b(c_N)$	for $\forall a \neq b \in \boldsymbol{M}$
Abs 2b Type 1-2	a. $\neg U_a(c_i) \wedge \neg\exists c_j\, U_a(c_j)$ b. $U_a(c_i) \wedge \neg\exists c_j\, U_a(c_j)$ c. $U_a(c_i) \wedge \exists c_j\, U_a(c_j)$	given $i \neq j \in \boldsymbol{N}$	for $\forall a \in \boldsymbol{M}$
Abs 2a Type 3-4	a. $\exists u_b[(\neg u_a \wedge \neg u_b)\ \mathbf{U}\ (u_a \wedge \neg u_b)\ \mathbf{F}\ u_b]$ b. $\exists u_b[(\neg u_a \wedge u_b)\ \mathbf{U}\ (u_a \wedge u_b)\ \mathbf{F}\ \neg u_b]$ c. $\exists u_b[(u_a \wedge \neg u_b)\ \mathbf{U}\ (\neg u_a \wedge \neg u_b)\ \mathbf{F}\ u_b]$ d. $\exists u_b[(u_a \wedge u_b)\ \mathbf{U}\ (\neg u_a \wedge u_b)\ \mathbf{F}\ \neg u_b]$ e. $\exists u_b[(\neg u_a \wedge \neg u_b)\ \mathbf{U}\ (\neg u_a \wedge u_b)\ \mathbf{F}\ u_a]$ f. $\exists u_b[(\neg u_a \wedge u_b)\ \mathbf{U}\ (\neg u_a \wedge \neg u_b)\mathbf{F}\ u_a]$ g. $\exists u_b[(u_a \wedge \neg u_b)\ \mathbf{U}\ (u_a \wedge u_b)\ \mathbf{F}\ \neg u_a]$ h. $\exists u_b[(u_a \wedge u_b)\ \mathbf{U}\ (u_a \wedge \neg u_b)\ \mathbf{F}\ \neg u_a]$	given $U_a(c_N), U_b(c_N)$	for $\forall a \neq b \in \boldsymbol{M}$
Abs 2b Type 3-4	a. $(\neg U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{U}\ (U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{F}\ U_a(c_j)$ b. $(\neg U_a(c_i) \wedge U_a(c_j))\ \mathbf{U}\ (U_a(c_i) \wedge U_a(c_j))\ \mathbf{F}\ \neg U_a(c_j)$ c. $(U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{U}\ (\neg U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{F}\ U_a(c_j)$ d. $(U_a(c_i) \wedge U_a(c_j))\ \mathbf{U}\ (\neg U_a(c_i) \wedge U_a(c_j))\ \mathbf{F}\ \neg U_a(c_j)$ given $i \neq j \in \boldsymbol{N}$		for $\forall a \in \boldsymbol{M}$

The third row finds Type 3-4 UCCAs using Abstraction 2a and produces an output consistent with Table 4-10. The notation employs Linear Temporal Logic (LTL) to describe the different sequences of starting and ending different control actions relative to one another.

Each equation is a sequence of three time periods. Item (a) in the list describes those three periods as they pertain to starting u_a before starting u_b. First, the initial condition treats both u_a and u_b as not provided because they have not yet started. This condition holds ***U**ntil* the second step, represented by LTL temporal operator **U** [207], when u_a is started and, therefore, is now provided. Finally, in some ***F**uture* third step, denoted by LTL operator **F**, u_b is started. As

discussed in Section 4.2.1, no assumption is made that u_a is still provided by the time this last step occurs.

The fourth row in Table 4-14 uses Abstraction 2b to output Type 3-4 UCCAs like those shown in Table 4-12. The cases in that row follow the same three temporal steps defined above in LTL. As such, item (c) in that row represents any one controller c_i ending u_a before any other controller c_j starts u_a.

Each enumerated control combination is encoded into a tuple, $\boldsymbol{C_x} = (\boldsymbol{C_x}, \boldsymbol{U_x}, \boldsymbol{T_x})$, as defined below. By abuse of notation, the abstracted set of multiple controllers or of multiple control actions can be encoded as any single element in $\boldsymbol{C_x}$ and $\boldsymbol{U_x}$. A superscript on $\boldsymbol{C_x}$ (i.e., $\boldsymbol{C_x^{Abs2a,T12}}$), informs which abstraction and type of UCCA is enumerated. For Type 3-4 UCCAs, the order of the elements in vectors $\boldsymbol{U_x}$ and $\boldsymbol{T_x^{34}}$ reflects the temporal sequence.

$\boldsymbol{C_x}$ ≔ vector of controllers involved in enumeration

$\boldsymbol{U_x}$ ≔ vector of control actions paired with $\boldsymbol{C_x}$

$\boldsymbol{T_x^{12}} = \{not\ provide, provide\}$; represents if elements $\boldsymbol{C_x}$ provide elements in $\boldsymbol{U_x}$

$\boldsymbol{T_x^{34}} = \{start, end, \emptyset\}$; applied by $\boldsymbol{C_x}$ to $\boldsymbol{U_x}$

Example: for the multi-UxS system, if $u_a = jam$, the third item of each row in Table 4-14 encodes $\boldsymbol{C_x}$ as Equations (8)-(11).

$$\boldsymbol{C_x^{Abs2a,T12}} = (\{team, team\} \quad \{[jam], [strike \vee track]\} \quad \{0,1\}) \tag{8}$$

$$\boldsymbol{C_x^{Abs2b,T12}} = (\{c_i, \exists c_j\} \quad \{[jam], [jam]\} \quad \{1,1\}) \tag{9}$$

$$\boldsymbol{C_x^{Abs2a,T34}} = (\{team, team\} \quad \{[jam], [strike \vee track]\} \quad \{end, start\}) \tag{10}$$

$$\boldsymbol{C_x^{Abs2b,T34}} = (\{c_i, \exists c_j\} \quad \{[jam], [jam]\} \quad \{end, start\}) \tag{11}$$

$\boldsymbol{C^{Abs}}$ denotes the set of all $\boldsymbol{C_x}$ created in the above enumeration. A MATLAB prototype developed to demonstrate this concept automatically produces four tables with the control combinations and placeholders for an analyst to enter potential contexts and other information described in the next step of the Algorithm. Examples of UCCA tables created using the automation are shown in Appendix 2.

Lines 2-4: Human-Identified Context of UCCAs

Once the automation has produced all the potential UCCAs, a human analyst determines for each one (Line 2) if there is a context, or multiple contexts, in which that control combination is hazardous (Line 3). As part of this, the analyst also traces each context to the hazard(s) the UCCA leads to. The context may describe the violation of safety constraints previously identified in the analysis. This part of the algorithm requires human expertise and intuition across multiple domains.

While some contexts can be formally described so that they are automatically found, it is challenging to scale that up beyond simple problems. In addition, the context may involve the environment the system interacts with, which is too unpredictable to fully describe formally [170].

The same concerns have limited the adoption of formal methods in certification (see Chapter 2.3). Humans, as creative and critical thinkers, are better suited for this step.

If a hazardous context is found, it is appended to the control combination (Line 4). For UCCAs defined using Abstractions 2a, the analyst also specifies vector $\boldsymbol{U_x^{rel}}$ to designate the control actions in u_a and u_b that are relevant to the context. Abstractions 2b UCCAs only include one control action and therefore do not require $\boldsymbol{U_x^{rel}}$ to be specified.

Lines 5-6: Automated Refinement of UCCAs

After the human specifies the context(s), the hazard traceability, and the relevant control actions, the automation reads in each unique abstracted UCCA as tuple $\boldsymbol{\mathcal{U}_x^{abs}} = \left(\boldsymbol{C_x}, \boldsymbol{U_x}, \boldsymbol{T_x}, context_x, \boldsymbol{U_x^{rel}}\right)$ (Line 5). The potential UCCAs that are repeated or not considered unsafe are removed from further consideration.

Example: for the multi-UxS system, the $\boldsymbol{\mathcal{U}_x^{abs}}$ encoded for abstracted UCCA 4 specified in Section 4.2.3 is shown in Equation (12). In this case, only the *track* and *strike* commands are relevant to the context.

$$\boldsymbol{\mathcal{U}_x^{abs}} = (\{team, team\} \quad \{[track], [jam \vee strike]\} \quad \{0,1\} \quad \{'when \ldots'\} \quad \{track, strike\}) \quad (12)$$

Next, Algorithm 1 calls a function to refine each abstracted UCCA (Line 6). Here, the automation finds every combination of specific controllers that can contribute to the collective control output in the abstracted UCCA.

For Type 1-2 UCCAs, the tool first considers all the possible control combinations $\boldsymbol{\mathcal{P}} = f(\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{A}, \boldsymbol{T})$ by listing every combination of every controller $\boldsymbol{N}$, with every control action $\boldsymbol{M}$ it has the authority $\boldsymbol{A}$ to provide it, and the options $\boldsymbol{T}$ for providing or not providing that action. Each combination is represented by tuple $\boldsymbol{\mathcal{P}_e} = \left(\boldsymbol{C_e}, \boldsymbol{U_e}, \boldsymbol{T_e^{T12}}\right)$ using the same format as $\boldsymbol{C_x^{T12}}$.

Example: for the multi-UxS system, one $\boldsymbol{\mathcal{P}_e}$ is shown in Equation (13). To shorten the equation, the *jam*, *strike*, and *track* commands are listed a *j*, *s*, *t* respectively. By abuse of notation, each controller in $\boldsymbol{C_e}$ is reiterated for each element in the vectors the controller is matched with with in both $\boldsymbol{U_e}$ and $\boldsymbol{T_e^{T12}}$. In other words, $\{UAS_1, UAS_2, robot\}$ should be interpreted here as $\{[UAS_1, UAS_1, UAS_1], [UAS_2, UAS_2, UAS_2], [robot, robot, robot]\}$.

$$\boldsymbol{\mathcal{P}_e} = (\{UAS_1, UAS_2, robot\} \quad \{[j,s,t], [j,s,t], [j,s,t]\} \quad \{[0,0,1], [0,1,0], [0,1,0]\}) \quad (13)$$

The function then looks for equivalence of the union of control actions provided by all controllers in $\boldsymbol{\mathcal{P}_e}$ to the relevant control actions specified in the abstracted UCCA $\boldsymbol{\mathcal{U}_x^{abs}}$**.** This step is simplified using the earlier assumption that the context of the UCCA is agnostic to which controllers issue the control actions. Equivalence can therefore be determined using subsets $\boldsymbol{\mathcal{P}_e'} = (\boldsymbol{U_e}, \boldsymbol{T_e})$ and $\boldsymbol{\mathcal{U}_x^{abs\prime}} = (\boldsymbol{U_x}, \boldsymbol{T_x})$, which remove the controllers, the contexts, and the irrelevant control actions from consideration. In simple terms, if a relevant control action is provided in the abstracted Type 1-2 UCCA, equivalence is achieved if any controller in the refined set provides that control action.

Example: for the multi-UxS system, Table 4-9 refines a UCCA in which the team provides ¬*track* and *strike*. In this case, any combination where no controller *tracks* and one or more controllers *strike* meets this criterion. As such, items 4.4-4.7, which include multiple controllers

providing the *strike* command, are considered equivalent to the collective team output. The associated $\boldsymbol{\mathcal{U}_x^{abs\prime}}$ and equivalent $\boldsymbol{\mathcal{P}_e'}$ defined for item 4.4 are listed in Equations (14)-(15).

$$\boldsymbol{\mathcal{U}_x^{abs\prime}} = (\{jam, strike\} \quad \{0,1\}) \tag{14}$$

$$\boldsymbol{\mathcal{P}_e'} = (\{[j,s],[j,s],[j,s]\} \quad \{[0,0],[0,1],[0,1]\}) \tag{15}$$

However, *special interactions* may also be defined to influence how equivalence is determined given the context of the UCCA. These interactions may reduce or expand the set of combinations that are considered equivalent. For instance, if multiple controllers provide the *jam* command, they may interfere with each other and yield a collective output equivalent to no *jam* being provided. In this case, if a high-level UCCA states the team provides ¬*jam* and *strike*, any refined UCCA that involves both UAS_1 and UAS_2 providing *jam* and any controller providing *strike* is treated as equivalent. This example would expand the set of combinations to consider.

The special interaction is defined as a tuple $\boldsymbol{\mathcal{S}} = (\boldsymbol{U}, \boldsymbol{T_{in}}, \boldsymbol{\Sigma}, \boldsymbol{T_{out}})$, where $\boldsymbol{\Sigma}$ specifies the rules that are applied to the control actions in $\boldsymbol{U}$ and whether or not they are provided $\boldsymbol{T_{in}}$. The rules indicate how to treat the collective output of those commands using items in vector $\boldsymbol{T_{out}}$. In the *jam* example above, $\boldsymbol{\Sigma}$ includes the rule that if the number of controllers providing the *jam* control action in $\boldsymbol{U}$ exceeds 1, treat *jam* collectively as $\boldsymbol{T_{out}^{12}} = not\ provide$.

Example: for the multi-UxS system, according to this special interaction, the $\boldsymbol{\mathcal{P}_e'}$ defined in Equation (16) would be considered equivalent to the $\boldsymbol{\mathcal{U}_x^{abs\prime}}$ previously listed in Equation (14).

$$\boldsymbol{\mathcal{P}_e'} = (\{[j,s],[j,s],[j,s]\} \quad \{[1,0],[1,1],[0,1]\}) \tag{16}$$

The output of this part of the algorithm is a set of refined control combinations $\boldsymbol{\mathcal{C}_x^{ref}}$ that are considered equivalent to the abstracted UCCA $\boldsymbol{\mathcal{U}_x^{abs}}$. $\boldsymbol{\mathcal{C}_x^{ref}}$ is evaluated using Equation (17) for Type 1-2 UCCAs, which is the mathematical representation of the concepts described above.

$$\boldsymbol{\mathcal{C}_x^{ref,T12}} = \bigcup_{\boldsymbol{e}} \boldsymbol{\mathcal{P}_e} \mid \left(\boldsymbol{\mathcal{U}_x^{abs\prime}} \equiv \bigcup_{i} \boldsymbol{\mathcal{P}_e'} \mid \boldsymbol{\mathcal{S}}, \forall i \in \boldsymbol{N} \right), \forall e \in \boldsymbol{\mathcal{P}} \tag{17}$$

Example: for the multi-UxS system, the Equation (18) shows $\boldsymbol{\mathcal{C}_x^{ref,T12}}$ represented by Table 4-9 and repeated in the discussion below in Table 4-15. The same abuse of notation used in Equation (13) applies.

$$\boldsymbol{\mathcal{C}_x^{ref,T12}} = \begin{vmatrix} (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,0],[0,0],[0,1]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,0],[0,1],[0,0]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,0],[0,0]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,0],[0,1],[0,1]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,0],[0,1]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,1],[0,0]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,1],[0,1]\}) \end{vmatrix} \tag{18}$$

Type 3-4 UCCAs are refined using a similar process. For Abstraction 2a, every possible combination of any one controller, with the proper authority, issuing each of the relevant control actions in the UCCA is enumerated. For Abstraction 2b, every possible combination of one controller issuing the earlier and the later common command signal is found. In both cases, each

enumerated item is labeled with *start* or *end* as designated by $\boldsymbol{T_x^{34}}$ in $\boldsymbol{\mathcal{U}_x^{abs}}$. The terms are rearranged into set $\boldsymbol{\mathcal{C}_x^{ref,T34}}$ using similar conventions as in $\boldsymbol{\mathcal{C}_x^{ref,T12}}$.

Example: for the multi-UxS system, Equations (19) and (20) represent, respectively, $\boldsymbol{\mathcal{U}_x^{abs}}$ and $\boldsymbol{\mathcal{C}_x^{ref,T12}}$ for the unsafe sequence that involves ending *jam* (j) before starting *strike* (s) and *track* (t). To shorten the representation, UAS_1, UAS_2, *robot*, *start* and *end* are listed as U_1, U_2, r, S and E respectively. Furthermore, the backets around the *jam* $[j]$ and its *end* $[E]$ or null $[\emptyset]$ are not shown, but are implied in Equation (20). The same abuse of notation described in Equation (13) applies in Equation (20).

$$\boldsymbol{\mathcal{U}_x^{abs}} = (\{team, team\} \quad \{[j], [s \vee t]\} \quad \{end\ (E), start\ (S)\} \quad \{'when \ldots'\} \quad \{j, s, t\}) \tag{19}$$

$$\boldsymbol{\mathcal{C}_x^{ref,T34}} = \begin{vmatrix} (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[S,S]],[\emptyset,[\emptyset,\emptyset]],[\emptyset,[\emptyset,\emptyset]]\}) \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[S,\emptyset]],[\emptyset,[\emptyset,S]],[\emptyset,[\emptyset,\emptyset]]\}) \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[S,\emptyset]],[\emptyset,[\emptyset,\emptyset]],[\emptyset,[\emptyset,S]]\}) \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[\emptyset,S]],[\emptyset,[S,\emptyset]],[\emptyset,[\emptyset,\emptyset]]\}) \\ \vdots & \vdots & \vdots \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[\emptyset,[\emptyset,\emptyset]],[\emptyset,[\emptyset,\emptyset]],[E,[S,S]]\}) \end{vmatrix} \tag{20}$$

In this work, Type 3-4 UCCAs do not employ special interactions $\boldsymbol{\mathcal{S}}$ to specify how different refined sequences of actions should be considered equivalent to the collective sequencing. Such interactions were not considered to be relevant to the problems analyzed as part of this research. However, if a need arises to include such special considerations, a version of Equation (17) that applies to Type 3-4 UCCAs could be derived. That is beyond the scope of this work.

Lines 7: Automated Pruning of Additional Equivalent Combinations

In the development of loss scenarios, which occurs later in the hazard analysis, causal factors are analyzed to explain how the control combinations in the UCCA may occur. This process needs the refined UCCAs to consider what controllers issued what control actions, as different controllers may have different causal factors. However, some of the controllers may be considered *interchangeable* from a scenario perspective. In such cases, the additional refined UCCAs that lead to the same causal analysis must be eliminated to avoid duplication of effort.

In the multi-UxS example (Figure 4-13), the engineering team may consider UAS_1 and UAS_2 to be interchangeable. Causal scenarios will be no different if it is UAS_1 that provides part of the UCCA, or instead, UAS_2. As a result, items 4.2 and 4.3 in Table 4-15 are equivalent as they both involve one UAS providing the *strike* command. No additional information is gained from 4.3 if 4.2 is analyzed. As such, UCCA 4.3 is pruned from the set.

Similarly, items 4.4 and 4.5 both involve one UAS and the robot providing the *strike* command and, therefore, 4.5 is pruned because it is duplicative. In contrast, items 4.2 and 4.5 are not equivalent even though they alternate the one UAS that provides *strike*. In item 4.2, UAS_2 provides the *strike* alone, while in 4.5, UAS_1 and the robot provide the *strike*.

Table 4-15. Pruning and Prioritizing Combinations in Refined UCCA 4

UCCA	Team	Team	Context
4	¬track	{~~jam~~ ∨ strike}	*when enemy must be tracked as strike occurs [H3]*

Refined	UAS_1	UAS_2	robot	Pruned?	Priority
4.1	¬track	¬track	strike	No	High
4.2	¬track	strike	¬track	No	High
4.3	strike	¬track	¬track	Yes (4.3 ≡ 4.2)	N/A - Pruned
4.4	¬track	strike	strike	No	Low
4.5	strike	¬track	strike	Yes (4.5 ≡ 4.4)	N/A - Pruned
4.6	strike	strike	¬track	No	Low
4.7	strike	strike	strike	No	Low

One of the inputs to Algorithm 1 is set $\boldsymbol{C^{int}}$ that contains z vectors of *interchangeable controllers*. In Line 7, a function uses this input to prune the equivalent duplicated control combinations from the set $\boldsymbol{C_x^{ref}}$ generated in Line 6. The process takes any two combinations $\boldsymbol{C_{x,m}^{ref}}$ and $\boldsymbol{C_{x,n}^{ref}}$ from $\boldsymbol{C_x^{ref}}$, checks if the vector of control efforts between any two interchangeable controllers are equivalent, and also checks that all control efforts by the other controllers are consistent. These conditions are captured in Equation (21), and if met, then the UCCAs are equivalent and $\boldsymbol{C_{x,n}^{ref}}$ is pruned from $\boldsymbol{C_x^{ref}}$.

$$\boldsymbol{C_{x,m}^{ref}} \equiv \boldsymbol{C_{x,n}^{ref}} \mid \left([U_m(c_i)] = [U_n(c_j)]\right) \wedge \left([U_m(c_j)] = [U_n(c_i)]\right) \wedge \left([U_m(c_k)] = [U_n(c_k)]\right), \quad \forall m \in \boldsymbol{C_m}, \forall n \in \boldsymbol{C_n}, i \neq j \in \boldsymbol{C_z^{int}}, \forall k \notin \boldsymbol{C_z^{int}}, \forall z \in \boldsymbol{C^{int}} \tag{21}$$

Example: for the multi-UxS system, UAS_1 and UAS_2 are interchangeable as defined by $\boldsymbol{C^{int}}$ in Equation (7), and one possible $\boldsymbol{C_x^{ref}}$ is represented by Equation (18). The two items $\boldsymbol{C_{x,m}^{ref}}$ and $\boldsymbol{C_{x,n}^{ref}}$, respectively shown in Equations (22) and (23), are members of $\boldsymbol{C_x^{ref}}$. They represent items 4.2 and 4.3 described in the same example above and shown in Table 4-15.

$$\boldsymbol{C_{x,m}^{ref,T12}} = (\{UAS_1, UAS_2, robot\} \quad \{[j,s],[j,s],[j,s]\} \quad \{[0,0],[0,1],[0,0]\}) \tag{22}$$

$$\boldsymbol{C_{x,n}^{ref,T12}} = (\{UAS_1, UAS_2, robot\} \quad \{[j,s],[j,s],[j,s]\} \quad \{[0,1],[0,0],[0,0]\}) \tag{23}$$

Here, $c_i = UAS_1 \in \boldsymbol{C^{int}}$, $c_j = UAS_2 \in \boldsymbol{C^{int}}$, $c_k = robot \notin \boldsymbol{C^{int}}$. The components of Equation (21) for this example are captured in Equations (24)-(27), and the three conditions necessary to prune $\boldsymbol{C_{x,n}^{ref,T12}}$ are met in Equation (28).

$$[U_m(c_i)],\ \forall m \in \boldsymbol{C_m} = [U_m(UAS_1)], u_m \in [jam, strike] = [0,0]; \tag{24}$$

$$[U_m(c_j)],\ \forall m \in \boldsymbol{C_m} = [U_m(UAS_2)], u_m \in [jam, strike] = [0,1]; \tag{25}$$

$$[U_m(c_k)],\ \forall m \in \boldsymbol{C_m} = [U_m(robot)], u_m \in [jam, strike] = [0,0]; \tag{26}$$

$$[U_n(c_i)],\ \forall n \in \boldsymbol{C_n} = [1,0];\ \ [U_n(c_j)],\ \forall n \in \boldsymbol{C_n} = [0,0];\ [U_n(c_k)],\ \forall n \in \boldsymbol{C_n} = [0,0]; \tag{27}$$

$$[U_m(c_i)] = [U_n(c_j)]; \qquad [U_m(c_j)] = [U_n(c_i)]; \qquad [U_m(c_k)] = [U_n(c_k)] \tag{28}$$

Conditions to specify controllers as *interchangeable* may vary by application. The taxonomy introduced in Figure 3-2 can help reason about what controllers lead to similar causal scenarios. It describes the structure of the interactions between controllers, which influences causality and the system dynamics. Commonality in some of the axes may help determine interchangeability.

As an example, consider the arbitrary collaborative system in Figure 4-15, in which five controllers share a process. The controllers are differentiable on at least the first two dimensions of the taxonomy. The interactions involved are human-human, human-machine, and machine-machine. The hierarchal structure includes supervisory control and peer interactions. Based on the controller types and hierarchy, the set of interchangeable controllers may be specified as $\boldsymbol{C^{int}} = \{\{human_1, human_2\}, \{machine_1, machine_2\}\}$. The other five dimensions in the taxonomy could also be included in this consideration if of interest to the analysis.

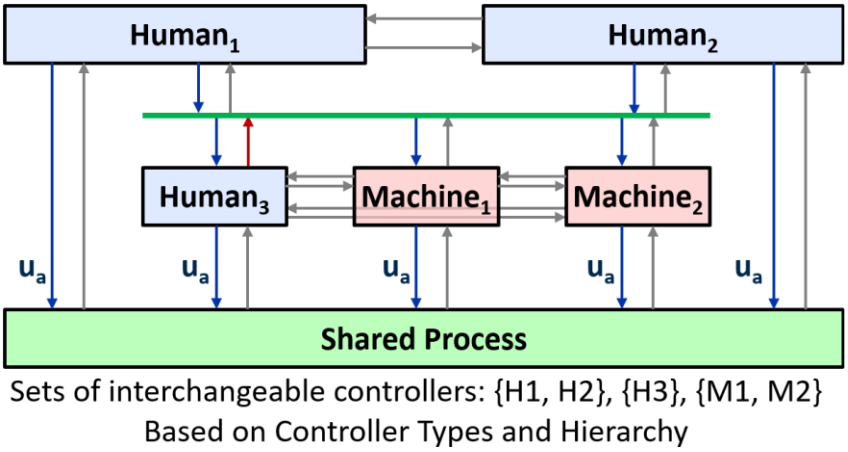

Figure 4-15. Interchangeable Controllers in Arbitrary Collaborative System

Lines 8-9: Automated Prioritization and Output of UCCAs

The refined UCCAs may optionally be prioritized to focus the remainder of the hazard analysis (Line 8). The goal is to highlight the UCCAs that will contribute more new information in causal scenario analysis and devalue those that provide more repetitive information. The prioritization scheme implemented in this dissertation is a heuristic formed on engineering judgment, but it could be implemented differently based on other application needs.

The concept is illustrated using again the ¬*track* and *strike* UCCA from the multi-UxS example refined in Table 4-15. Scenarios for this UCCA must explain why (1) any controller would provide *strike* when (2) no controller provides *track*, and vice versa. It is arguably less important in the context of the UCCA to focus on why multiple controllers would provide *strike*. As such, in the remaining set of refined UCCAs that were not pruned, those with one controller providing *strike* are prioritized over those with multiple controllers issuing that command.

It is also important to note, in this example, that if there is a context in which multiple controllers providing *strike* is unsafe, that interaction is specifically addressed in Abstraction 2b, where combinations of controllers provide the same action (e.g., $c_i\ strike\ \wedge\ \exists c_j\ strike$). For this reason, the prioritization scheme does not devalue combinations of two controllers providing the action in Abstraction 2b. However, using a similar logic as above, it does deprioritize combinations of three or more controllers providing the same action, as those may be repetitive.

In this dissertation, the prioritization scheme is also employed to devalue UCCAs in which all the control actions are provided by one controller. For example, if commands u_1 and u_2 are unsafe together, and if controller c_1 is the only one to provide those two commands, the UCCA is deprioritized. Such instances are arguably less related to collaborative control. However, other applications may choose to treat such occurrences with higher priority.

The special interactions specified in $\boldsymbol{S}$ can also be used to influence prioritization. In the example previously used to illustrate $\boldsymbol{S}$, multiple controllers providing the *jam* command are treated collectively as not providing *jam*. Here, $\boldsymbol{S}$ ensures that cases of two controllers providing *jam* in a UCCA are not deprioritized. However, by the same reasoning provided for Abstraction 2b UCCAs above, those with any three controllers providing *jam* are deprioritized.

Because the assumptions underpinning prioritization are softer than those used in the pruning process, the deprioritized UCCAs are not eliminated. UCCAs are presented to the analyst by order of priority, and the option remains to analyze those labeled as lower in priority in scenario development.

The set of UCCAs refined, pruned, and prioritized using automation is returned by Algorithm 1 as set $\boldsymbol{U^{ref}}$ (Line 9). An example of the output produced by the prototyped automation tool is provided in Appendix 2. The UCCAs, both abstracted and refined, are now ready for the analyst to proceed to the last step in the hazard analysis: the development of causal scenarios.

4.3 Causal Scenarios in Collaborative Control

The fourth and final step in STPA develops loss scenarios to identify causal factors that can lead to the unsafe control actions (UCAs) [50]. Safety constraints can then be specified to guide the design and operation of a system to eliminate or control these factors to prevent losses.

In STPA, scenarios are identified by analyzing each UCA to determine (1) why the controller would provide the UCA and (2) why a control action would be improperly executed or not executed leading to the outcome of the UCA. The process explores potential breakdowns in four different parts of a feedback control loop. The STPA guidance describes common factors to consider in each part to assist in the analysis (see Figure 4-16).

An approach proposed by Thomas aims to add structure and enhance the ability to build scenarios top-down [208]. The process starts by formulating four *basic scenarios*, generically listed below, which originate from the four parts of the feedback control loop. The basic scenarios are then further refined as necessary to develop safety constraints to mitigate the associated factors.

1. Basic Scenario 1: Unsafe Controller Behavior: the controller receives adequate feedback, but still makes unsafe decisions.
2. Basic Scenario 2: Unsafe Feedback or Other Information: the controller receives inadequate feedback leading to an unsafe decision.
3. Basic Scenario 3: Unsafe Control Path: the controller provides a safe control action, but the controlled process receives a control action that is unsafe.
4. Basic Scenario 4: Unsafe Process Behavior: the safe control action is received by the controlled process, but the process behaves in an unsafe way. [208]

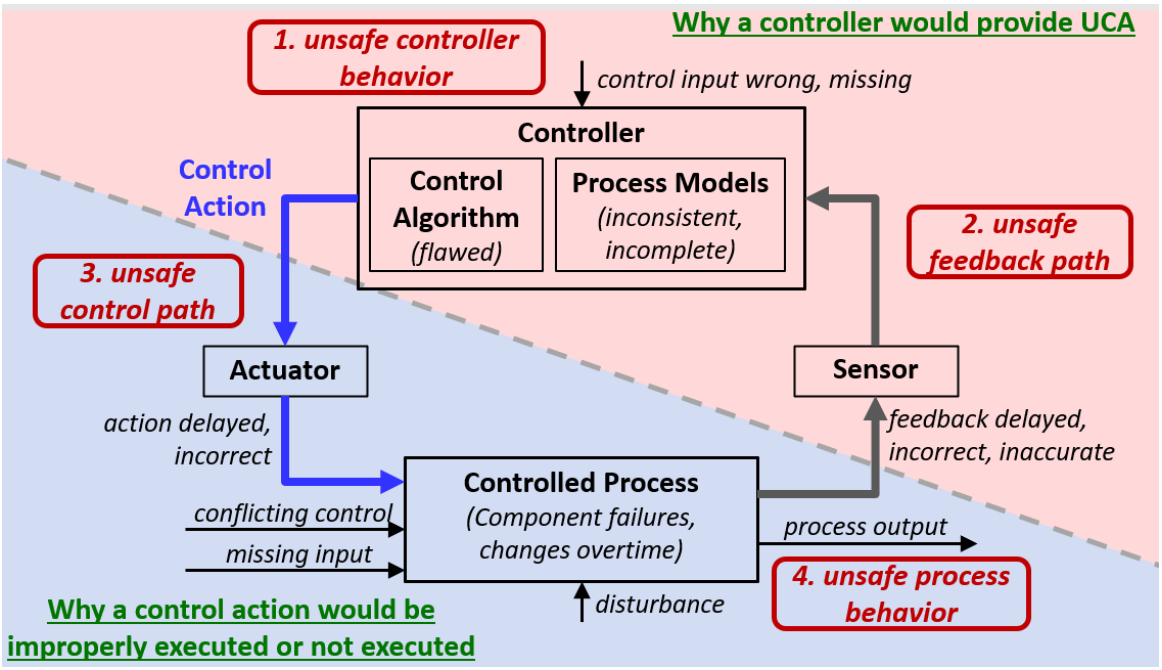

Figure 4-16. Areas of Potential Breakdown in a Feedback Control Loop (derived from [50])

The scenario development process in STPA is able to uncover causal factors not found by other hazard analysis techniques [209]. However, the approach focuses on one unsafe control action provided by one controller at a time. As described in Section 4.2, some causal factors may only be identified by exploring how multiple control actions are unsafe together.

Now that unsafe combinations of control actions (UCCAs) are identified, a process outlined in Figure 4-17 is introduced to develop causal scenarios from these UCCAs. The process has the following goals. First, it aligns with STPA by considering both (1) why unsafe control actions would be provided and (2) why control actions would not be properly executed. Second, it provides a mechanism to analyze the multiple feedback control loops involved in the UCCAs collectively. Third, it accounts for the collaborative control dynamics defined in Chapter 3. And fourth, the approach is systematic in scenario identification and refinement.

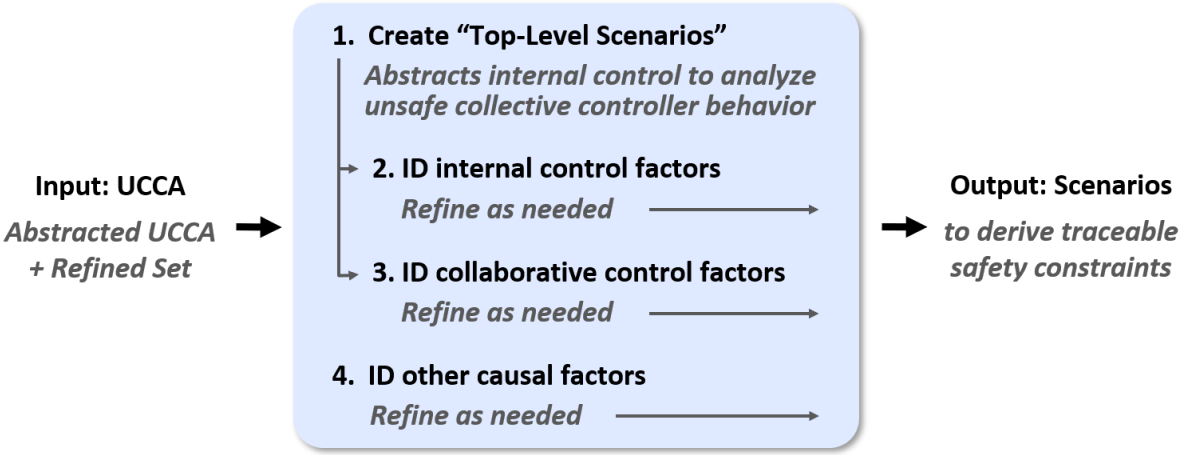

Figure 4-17. Process to Develop Causal Scenarios from a UCCA

The input to the process is a UCCA identified using the technique in Section 4.2. Each UCCA represents an unsafe collective control output by the team to the shared process. As in STPA, the scenario identification process explores factors that lead to unsafe behaviors of the team of controllers, its feedback paths, its control paths, and the controlled process. However, this work emphasizes how the interactions among the controllers on the team contribute to unsafe collective team behavior. The analysis of the feedback paths, control paths, and controlled processes follows the same reasoning as in STPA. This key idea is illustrated in Figure 4-18.

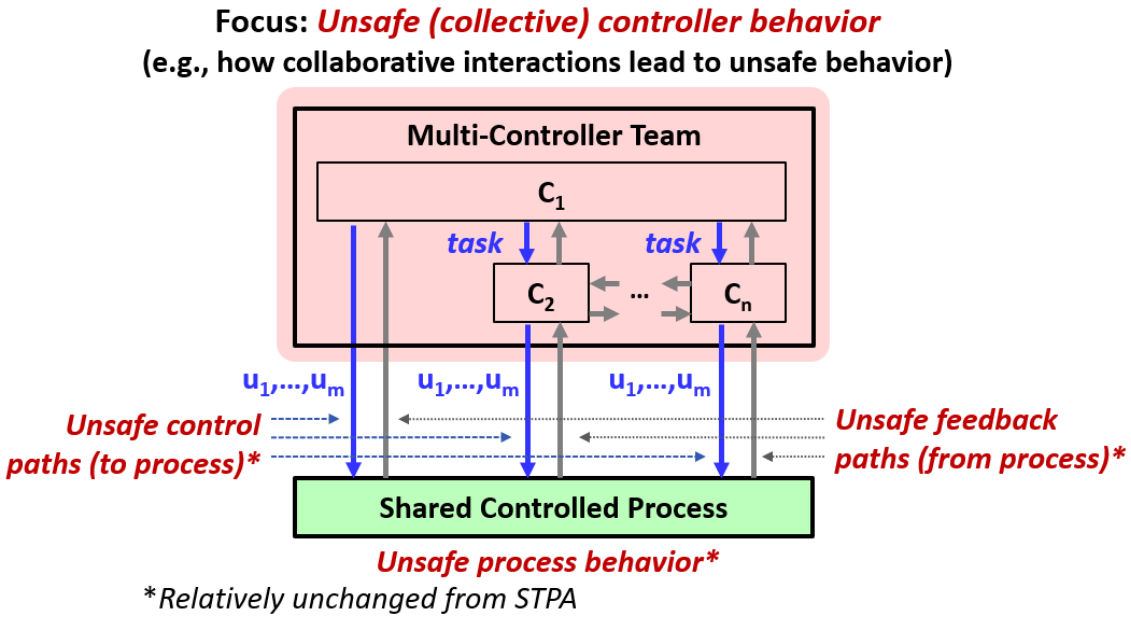

Figure 4-18. Four Areas of Potential Breakdown in Multiple Feedback Control Loops

In Step 1, the scenario identification process (Figure 4-17) explores how hierarchal control within the team contributes to the unsafe collective controller behavior in the UCCA. To illustrate this concept, consider the system in Figure 4-18, where C_1 has control authority over the other controllers on the team in addition to the shared process. The first step considers how the different potential control actions from C_1 to the other controllers relate to the UCCA.

In simple terms, this step investigates how the UCCA could occur if, for example, C_1 commands the team to provide an unsafe output, or as another example, if the other controllers do not properly execute commands from C_1. Each example represents a new scenario to consider. A method defined in Section 4.3.1 helps to systematically account for the different possible internal control actions that lead to unsafe collective team behavior using *top-level scenarios*.

Each of these scenarios is then iteratively refined using a template introduced in Section 4.3.2. In Step 2, the template finds causal factors in the control loops internal to the team. In Step 3, the template identifies factors related to the collaborative control dynamics of the team.

Finally, Step 4 identifies factors that relate to unsafe feedback paths from the controlled process, unsafe control paths to the controlled process, and unsafe controlled process behavior. These items follow the same approach as used in STPA.

The output of the process is a set of causal scenarios from which engineers can derive safety constraints to eliminate or mitigate the factors that lead to hazards. The remainder of the section

describes each step of the process in more detail. Examples of its application and of the safety constraints that are derived from it are provided in the Chapter 5 case study.

4.3.1 Step 1: Top-Level Scenarios to Reason about Internal Control

Step 1 of the scenario identification process examines how the collective output in a UCCA relates to the different possible control actions internal to the team. To illustrate this concept, assume that the system shown in Figure 4-18 has a UCCA that involves multiple controllers providing control action u_1. This case occurs in the multi-UxS example in Section 4.2, when multiple controllers provide the *jam* command (see Table 4-11), and it is a Type 1-2 UCCA.

Using the process described in Section 4.2.4, the two priority refined UCCAs, in this case, are (1) c_1 and c_2 both provide u_1, and (2) c_2 and c_n both provide u_1. Step 1 now explores how the commands provided by c_1 to $\{c_2, \ldots, c_n\}$ may have contributed to these outcomes. Assuming c_1 can task the other controllers to execute u_1, Figure 4-19 enumerates the four different possible c_1 internal control actions that are relevant to each of the two refined UCCAs.

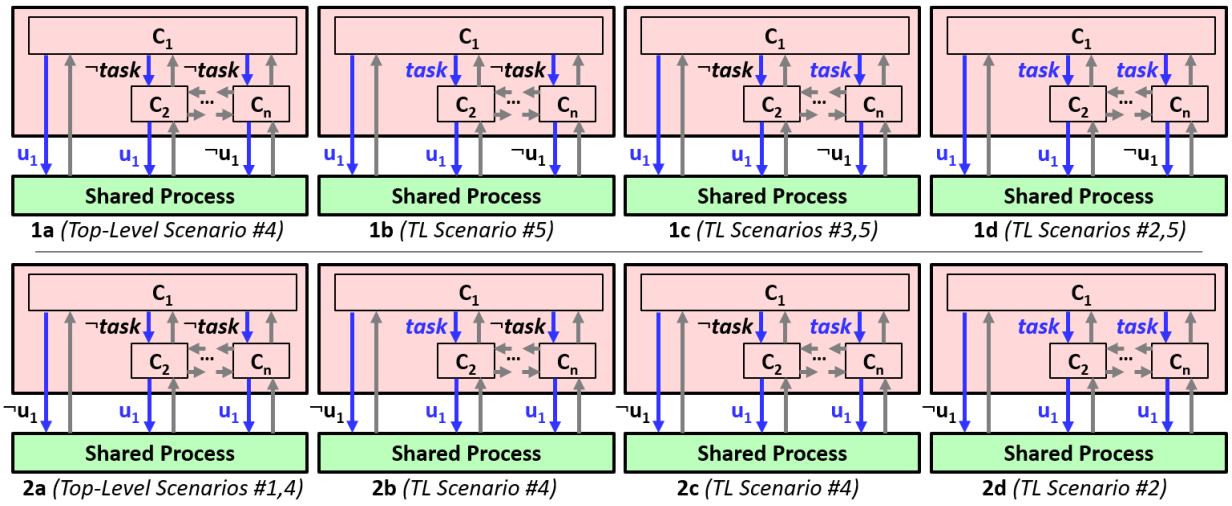

Figure 4-19. Possible Internal Control Actions that Can Lead to Type 1-2 UCCA

Each different set of internal control actions represents a new potential scenario to analyze. For instance, in item 1a, c_1 does not command any other controller to provide u_1, and instead, provides the command itself. Yet, c_2 also issues the command despite not being tasked. This forms a scenario that can be further refined to explain the unsafe behavior of the team as a whole.

Reasons for this outcome may include c_1 unintentionally providing the task, c_2 receiving the command from another controller, c_2 unintentionally providing the command, and so on. A systematic process to explore these causal factors is introduced in the next section. The other items in Figure 4-19 may also be developed into additional scenarios.

While this simple example only involves one type of control action (u_1), it still results in eight potential scenarios to further analyze. This number grows exponentially with the number of different types of internal control actions involved $\{u_1, \ldots, u_n\}$. Therefore, a full enumeration with any more complexity can produce an intractable number of scenarios. This is similar to the problem encountered in UCCA identification. Simplification is again necessary.

Abstraction is again applied to manage the combinatorial complexity. The different possible internal control actions are abstracted into *top-level* scenarios that cover their general concern toward the unsafe collective output. Table 4-16 lists the generic *top-level* scenarios that help reason about how internal control relates to Type 1-2 UCCAs. These scenarios are directly traceable to UCA Types 1 and 2 in STPA and have a similar intent to the *basic scenarios* by Thomas [208].

Table 4-16. Top-Level Scenarios that Address Internal Control Issues for Type 1-2 UCCAs

#	Top-Level Scenario	Full Top-Level Description
1	Direction Not Provided (Unsafe)	A controller does not direct other controllers on the team as necessary for the team to execute safe collective control of the shared process.
2	Direction Provided (Unsafe)	A controller directs other controllers on the team in a way that leads to unsafe collective control. Includes: directing wrong controller to provide command, directing multiple controllers in a way that conflicts with one another, and directing controller to provide incorrect command
3	Direction Provided (Safe) but Not Executed Properly (Unsafe)	A controller directs other controllers on the team adequately, but some of those controllers do not execute directions properly, which leads to unsafe collective control. Includes: controllers do not provide some commands, controllers provide commands improperly, wrong controller provides command
4	Direction Not Provided (Safe) but Executed (Unsafe)	A controller adequately does not direct other controllers on the team to provide certain commands, but some of those controllers provide them anyways, which leads to unsafe collective control.
5	Controller Actions to Process and Directions it Provides (Unsafe)	A controller provides control actions to the shared process that are unsafe in combination with how it directs other controllers on the team. Includes: improperly providing a control action that is necessary in combination with directed actions, providing a control action that conflicts with directed actions

While the top-level scenarios are designed to provide coverage over the different possible internal control combinations, they are not mutually exclusive of one another. Figure 4-19 illustrates this point by mapping each control combination into these scenarios. Some of the cases fit into multiple scenarios. The analytical overlap is intentional to avoid potential gaps.

For example, in item 1c, c_1 tasks a specific controller to provide u_1, but a different controller provides it instead. This situation fits into top-level scenario #3 in Table 4-16. In addition, despite tasking another controller to provide u_1, c_1 also provides the command itself, which is captured by top-level scenario #5. The scenarios help systematically consider these issues and are then further refined in the context of the UCCA, as described in the next subsection.

As defined in Chapter 3, *dynamic hierarchy* occurs when multiple controllers can mutually command each other. As such, *dynamic hierarchy* contributes to the possible control actions

internal to the team. For this reason, this interaction is addressed in the causal analysis using the top-level scenarios defined above. Multiple instances of each of the scenarios can be created to cover changes in the controller, providing directions to others on the team.

For example, the Pilot - Digital Copilot system surveyed in Chapter 3.3 exhibits dynamic hierarchy in collaborative checklist execution. Top-Level Scenario #1 can explore how the pilot does not direct the automation to execute certain functions and vice-versa, it can also examine how the automation does not direct the pilot. Examples of scenario development related to dynamic hierarchy are provided in the case study in Chapter 5.

Type 3-4 UCCAs, which describe how starting and ending control actions relative to one another are unsafe, face similar combinatorial challenges. For example, consider an unsafe gap in the handoff of control action u_1 between any two controllers in the same system analyzed above. In other words, c_i ends u_1 before c_j starts u_1.

This UCCA refines into the three priority UCCAs shown in Figure 4-20. Each may be related to different command options provided by c_1. In the figure ***F*** is the temporal operator for *some **F**uture step* and designates the later of the control action(s) started (S) or ended (E) in the sequence.

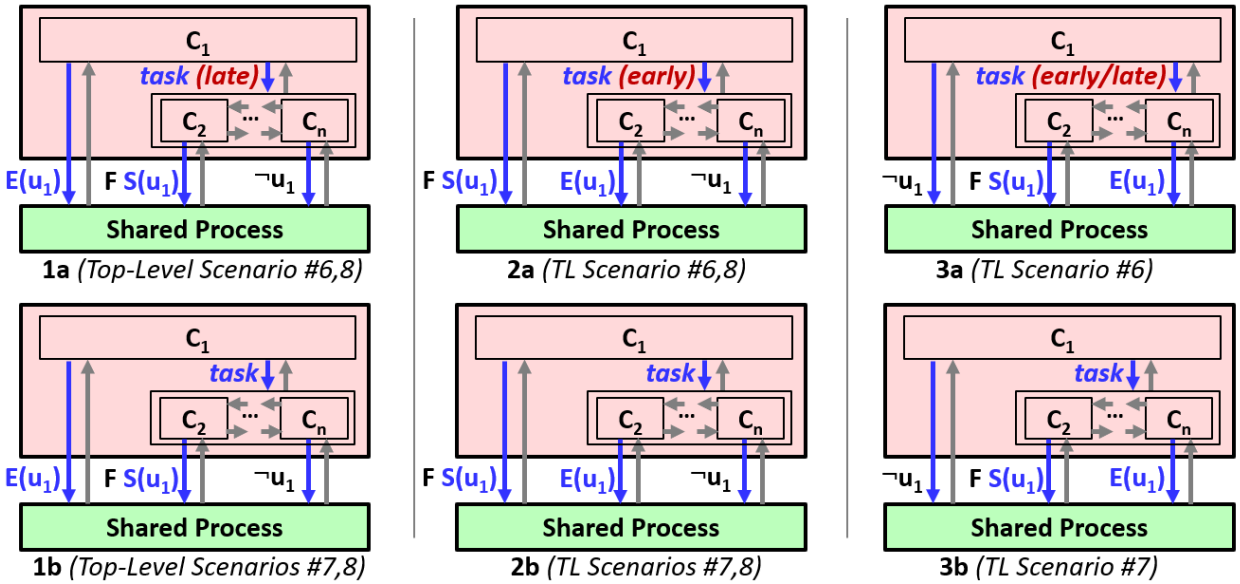

Figure 4-20. Possible Internal Control Actions that Can Lead to Type 3-4 UCCA

Using the same reasoning applied to Type 1-2 UCCAs, Table 4-17 provides the top-level scenarios to abstract the different possible internal controls that lead to Type 3-4 UCCAs. A demonstration of how to tailor top-level scenarios to analyze a system is provided in Chapter 5 for the case study. The next subsection explains how to refine these scenarios in the context of the UCCA.

Table 4-17. Top-Level Scenarios that Address Internal Control Issues for Type 3-4 UCCAs

#	Top-Level Scenario	Full Top-Level Description
6	Directed Sequence Unsafe	A controller directs other controllers on the team in a way that leads to unsafe temporal sequencing.
7	Directed (Safe) but Executed in Unsafe Sequencing	A controller adequately directs other controllers on the team, but the way in which those controllers execute the directions leads to unsafe temporal sequencing.
8	Controller Actions to Process and Directions Unsafe in Sequencing	A controller on the team provides control actions to the shared process that are unsafe in temporal sequencing with how it directs other controllers on the team.

In some systems, the set of collaborating controllers that share authority over a process are all peers and do not have authority to provide control actions to one another. In such cases, top-level scenarios 1-4 and 6-7 are still applicable to explore the different possible combinations of control actions provided to these controllers by supervising controller(s). Top-level scenarios 5 and 8 do not apply if the supervisor(s) do not issue control commands directly to the shared process. The multi-UxS system in Figure 4-13 illustrates this structure, where the UAS and the robot collaborate as peers, and the operator is the supervisor.

4.3.2 Step 2: Internal Control Causal Factors

Steps 2 and 3 in the extended scenario identification process identify causal factors that lead to the unsafe collective controller behavior in each top-level scenario. The template introduced in Figure 4-21, which is a refinement of the process overview in Figure 4-17, provides a systematic approach to consider these causal factors at a high level and then iteratively refine them as necessary. This section and the next introduce the key concepts in this template. Its application is demonstrated in the Chapter 5 case study.

Step 2 of the scenario identification is illustrated by the top yellow box in Figure 4-21. It involves finding, for each top-level scenario, the causal factors associated with feedback control loops internal to the team. The control loops explored are those where a controller provides control actions to the other controllers on the team. In the example shown in Figure 4-19, these are the feedback control loops from c_1 to c_2 and c_1 to c_n.

This step explores how the controller providing the direction (c_1 in the example) may have *unsafe control inputs*, an *inadequate control algorithm*, *inadequate models* of the controllers it is directing ($c_2, \dots, c_n$), and *unsafe control paths* to those controllers. These are the same factors considered in STPA [50]. The only difference is that multiple internal control loops may be considered at once.

The lower yellow box is a refinement template for one of the high-level causal factors. If more detail is needed to explain why the controller providing direction has an inadequate process model of the directed controller, the template points to different reasons for inadequate feedback. For example, c_2 may not send feedback to c_1, or c_1 may not receive the feedback, or c_1 may interpret the feedback incorrectly, and so on. Once again, this guidance is from STPA [50].

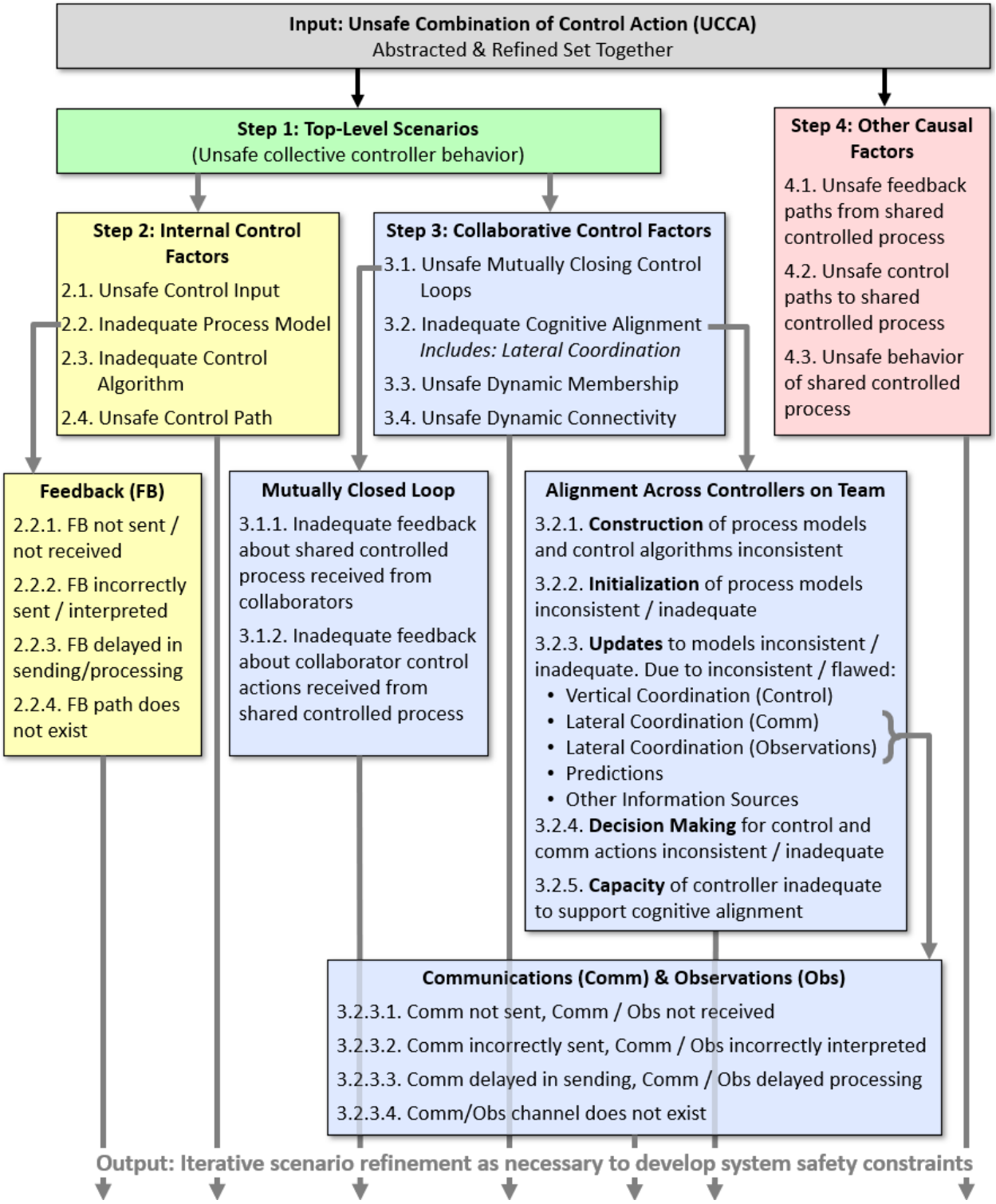

Figure 4-21. Iterative Refinement Template for Causal Scenario Development

4.3.3 Step 3: Collaborative Control Causal Factors

A key goal of this research is to extend the analysis to cover the collaborative control dynamics defined in Chapter 3. Up to this point in the analysis, four of the nine collaborative interactions have already been addressed. As explained in Section 4.2, *shared authority*, *dynamic authority*, and *transfer of authority* are inherently included in the UCCA identification. Similarly, *dynamic hierarchy* is captured in the top-level scenarios that describe internal control (see Section 4.3.1).

The five remaining collaborative control dynamics are addressed in Step 3 of the scenario development process, as illustrated in the top blue box in Figure 4-21. These include *mutually closing control loops*, *cognitive alignment*, *lateral coordination*, *dynamic membership*, and *dynamic connectivity*. The following describes how each is handled.

Mutually Closing Control Loop Causal Factors

In *mutually closing control loops*, feedback control loops are closed across multiple controllers. For scenario development, it is useful to refocus the control structure on this interaction. If in the example system above in Figure 4-18, u_1 and u_2 involve mutually closing loops, Figure 4-22 represents the refocused control structure to explore this dynamic, where "FB" means feedback.

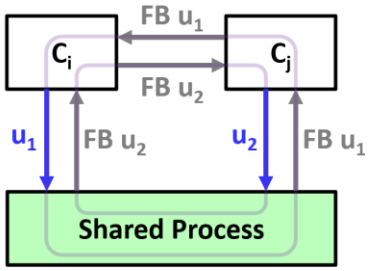

Figure 4-22. Refocused Control Structure for Mutually Closing Control Loops

In the refocused control structure, the controllers are generic (c_i and c_j) to account for different controllers involved in the refined UCCAs. The hierarchy between controllers does not need to be shown, as it is not the focus of this dynamic. The key concept to represent is that c_i senses feedback from a control action provided by c_j, and passes that feedback to c_j, which may otherwise not have access to it. This feedback also influences how c_i controls its part of the process and interactions with c_j.

A scenario refinement template is provided in Figure 4-21 (see the *Mutually Closed Loop* blue box) to further explore causal factors associated with such control loops. The analyst first considers the consequences of a controller receiving inadequate feedback from collaborators regarding its control actions to the shared process. In Figure 4-22, c_j may provide u_2 in an unsafe way if the feedback from c_i is inadequate.

Next, the analyst examines how a controller is influenced by inadequate feedback from the shared process regarding the actions of its collaborators. In the example, c_i may provide u_1 in an unsafe way if inadequate feedback from the process leads it to misinterprets how c_j is controlling the process. Reasons for the inadequate feedback can then be further refined using the same bottom yellow refinement template in Figure 4-21 previously discussed.

Cognitive Alignment and Lateral Coordination Causal Factors

In the *cognitive alignment* dynamic, scenario development focuses on why multiple controllers have process models and make decisions that are inconsistent with one another. A refinement template was developed using ideas synthesized from Thomas [210], France [189], and Johnson [191]. This template is shown in Figure 4-21 (see the *Alignment Across Controllers* blue box). To explore the causal factors in it explores, it is helpful to consider some of the items tracked in the process models of each controller as previously described in Figure 4-6.

The template first accounts for potential differences in how the cognitive functions were constructed. For example, two machines working together may be loaded with different software versions that make them incompatible with one another. Similarly, humans may have been trained differently than their teammates. These same concerns apply to human-machine teams.

The template then studies how process models may be initialized inconsistently across a team. Even if the controllers are aligned in cognitive construction, the different initial conditions available to each can prevent their models from synchronizing. For example, in the Air France 447 accident (see Chapter 1.2), the captain returned from crew rest after the initial aircraft control disturbance [35]. His mental model of aircraft control at that moment was initialized differently than the other two crew members, which contributed to the confusion in the cockpit.

The analysis then examines how the models of the collaborative controllers may be updated inconsistently with one another. Even if the cognitive functions are constructed and initialized similarly, flawed interactions may cause them to drift from one another over time. Several items are considered here. One is inconsistent *vertical coordination* as defined by Johnson [191]. The synchronization of models between multiple teammates may be influenced using hierarchal control. This occurs, for instance, when an Air Traffic Controller (ATC) provides traffic advisories to two merging aircraft.

The next items include communications and observations that lead to flawed *lateral coordination*. These relationships, which are expressed in the collaborative control structure, represent how controllers deliberately and non-deliberately influence each other without using control. For example, pilots of multiple aircraft at an uncontrolled airport laterally coordinate via active communication and passive observations. Inadequate exchange and interpretation of this information can lead models to drift. A template to further refine these factors is shown in the bottom blue box in Figure 4-21.

Here, lateral coordination is included as part of the broader cognitive alignment process because this model fits most of the aerospace systems studied in Chapter 3.3. However, if a system includes lateral coordination but not cognitive alignment, as occurs in some cases, the analyst can still use the lateral coordination refinement template shown in Figure 4-21 separately.

Models may also update inconsistently due to flawed *predictions* made by controllers regarding each other. Controllers often project what the future state of a process will be based on previous knowledge. Many machines employ such predictions in state estimation techniques to overcome noisy, infrequent, or missing feedback [211]. Humans use predictions to estimate where a teammate will be in the future to pass the ball or to determine how a pilot will navigate under lost communications [212]. Flawed predictions can contribute to model drift, especially in the absence of other information.

Lastly, misaligned model updates may result from inconsistencies in other sources of information. These may include differences in observations of common objects, as specified by Johnson [191]. They can also be misaligned feedback received from the shared controlled process or differences in the information received from the environment or controllers beyond the team.

The template for cognitive alignment then examines why *decision-making* may be inadequate as a team. This can occur even if the controllers are synchronized in cognitive construction, model initialization, and model updates. Controllers on a team may have consensus on how they will make a decision, but if it takes them too long to reach that decision or if the decision they collectively reach is incorrect, their output may be unsafe. An example of this is algorithmic churn in distributed systems, when controllers try to reoptimize too frequently based on each other's actions and ultimately do nothing [145].

Finally, the template considers the impact of controller *capacity* on cognitive alignment. As explained in Section 4.1, a controller may be limited by its workload and capabilities. It may not have the capacity to follow commands, coordinate, and make observations, predictions, and decisions that are synchronized with others.

Dynamic Membership and Dynamic Connectivity Causal Factors

The last two collaborative control dynamics shown in Figure 4-21 must be examined. *Dynamic membership* simply explores how a UCCA could occur because controllers come and go on the team. For example, if a controller takes on a task but that controller is subsequently removed from the team, it could affect the actions of collaborators and the collective output. The addition of a new controller or the uncertainty in the team composition may also be causal factors in unsafe control.

Dynamic connectivity considers how expected changes in the team topology can lead to the unsafe collective behavior. This factor is already covered, to some extent, in STPA by examining unsafe control paths and unsafe feedback paths that inhibit information flow. This work extends the consideration to include flawed information relaying, asynchronous information propagation, and temporary disconnects across the team.

Because these two dynamics are conceptually simpler than the others, no dedicated refinement templates were deemed necessary for them in this research. However, such templates can be created and added to the process as necessary in future work.

4.3.4 Step 5: Other Causal Factors

Steps 1, 2, and 3 in the scenario development process aim to explain the unsafe controller behavior of the collective team. As shown in Figure 4-18, the process also considers the other elements in the feedback control loop examined in STPA, including *unsafe feedback paths* from the shared controlled process, *unsafe control paths* to that process, and *unsafe process behavior*.

The analysis here follows the STPA guidance. A controller on the team may have an unsafe model of the shared process because of inadequate feedback it receives from the process. A controller may unintentionally provide or not provide a control action due to a flaw in the control path. Finally, the shared process may adequately receive the collective control inputs from the team but still exhibit unsafe behavior due to other inputs and disturbances.

Some of these factors may already be identified in the scenarios involving collaborative control, which is not a problem. The analytical process favors overlap instead of gaps in consideration.

4.3.5 Final Thoughts on Iterative Refinement Process

The scenario refinement process is demonstrated in the Chapter 5 case study, and a full dataset is available in Appendix 3 and 4. As in STPA, the development of causal scenarios in this work depends on human reasoning and leverages the experience and creativity of the analysis team. The refinement template aims to reduce some of the cognitive burden on human analysts by systematically directing their thought processes across the various factors.

However, the template should not be used as a checklist. The analysts cannot simply hit every item on the list and assume the analysis is complete. Some factors in the template may not be relevant, and others may require careful consideration at multiple levels of abstraction. Furthermore, no claim is made that the template is complete. Other factors beyond those listed may also be uncovered.

Finally, the determination of how much refinement is necessary to complete the analysis remains an open research question. The process encourages iterative refinement so that the high-level scenarios provide as complete coverage as possible. As shown in Chapter 6, the results of the hazard analysis can influence designers to remove complexity from the design to eliminate scenarios at a high level. However, in other cases, the analysis must provide further details to build safety into the design. How detailed is detailed enough varies on a case-by-case basis.

4.4 Summary of Extended Hazard Analysis

This chapter introduced techniques to fill a critical gap in the ability to conduct hazard analysis on systems that exhibit the complex interactions defined in Chapter 3. Prior to this work, no known method was available to systematically and rigorously analyze system safety for collaborative control systems.

As described in Chapter 2, the common hazard analysis techniques employed in aerospace, such as FHA, FMEA, FMECA, FTA, and HAZOP, are based on *Linear Chain-of-Events* causality models, which do not consider cyclic influence. Their very foundation makes them unable to analyze collaborative relationships involving mutual influence between controllers. Most of these methods also decompose the system into individual components, which fundamentally overlooks the interactions that occur in collaboration. Finally, these techniques are hardware focused and are unable to analyze causal factors related to human and software control effectively, much less when multiple humans and automated controllers work together.

The System Theoretic model in STAMP, on which STPA is built, is able to overcome these limitations, as explained in Chapter 2. However, STPA does not specifically address eight of the nine collaborative control dynamics. This makes the technique vulnerable to omitting these interactions from consideration or to ambiguity in how to handle them. The only exception is *dynamic connectivity*, which is explored, to some extent, in the analysis of unsafe feedback paths

and unsafe control paths in STPA [50]. In addition, a past STPA extension does address *lateral coordination* [191], but it does not comprehensively explore the other collaborative dynamics.

For this reason, three extensions, collectively known as STPA-Teaming, were developed to provide a capability to analyze safety in collaborative control systems. First, the *generic collaborative control structure* provides a mechanism to express collaborative interactions in STAMP models so that they are explicitly considered in the hazard analysis.

Second, the identification of *Unsafe Combinations of Control Actions (UCCAs)* provides a systematic approach to analyzing joint control contributions from multiple controllers. The system-theoretic foundation employs abstraction to manage the combinatorial complexity as needed to remain practical for real-world systems. The method maintains the rigor of STPA as it was derived from the specification of Unsafe Control Actions (UCAs). However, the extended formulation directly covers the dynamics of *shared authority*, *dynamic authority*, and *transfer of authority*.

Finally, the extended scenario development process provides a structured method to analyze causal factors that lead to a UCCA. The approach focuses on how the interactions between the multiple controllers on the team can contribute to unsafe collective team behavior. Furthermore, it provides a mechanism to specifically consider the causal influence of the remaining six collaborative control dynamics. Finally, the process emphasizes defining scenarios at a high level and iteratively refining them to the level required to develop safety constraints. Table 4-18 summarizes how STPA-Teaming analyzes causality associated with each of the nine collaborative control dynamics.

Table 4-18. Where Collaborative Control Dynamics are Analyzed in STPA-Teaming

Collaborative Control Dynamics	Causal relationships covered by
Shared Authority	UCCA identification (Section 4.2)
Dynamic Authority	UCCA identification (Section 4.2.2)
Transfer of Authority	UCCA identification (Section 4.2.2)
Dynamic Hierarchy	Top-level scenario identification (Section 4.3.1)
Mutually Closing Control Loops	Scenario refinement (Section 4.3.2)
Cognitive Alignment	Scenario refinement (Section 4.3.2)
Lateral Coordination	Scenario refinement (Section 4.3.2)
Dynamic Membership	Scenario refinement (Section 4.3.2)
Dynamic Connectivity	Scenario refinement (Section 4.3.2)

The inherent shortfalls identified in the common hazard analysis techniques and the explanation of why and how STPA was extended in this chapter support Hypothesis 2 of this dissertation.

> *Hypothesis 2: The system-theoretic collaborative interactions framework describes component interactions that are not specifically addressed by existing hazard analysis techniques, including STPA.*

The next chapter demonstrates the application of the extended hazard analysis on a real-world system concept and evaluates the performance of the technique compared to baseline STPA. Appendices 2-4 include the case study data that was produced using the methods developed in this chapter.