FSTPA-I: A Formal Approach to Hazard Identification via System Theoretic Process Analysis

Philip Asare
Charles L. Brown Department of Electrical and Computer Engineering
Department of Computer Science
University of Virginia,
Charlottesville, VA USA 22904
pka6qz@virginia.edu

John Lach
Charles L. Brown Department of Electrical and Computer Engineering
University of Virginia,
Charlottesville, VA USA 22904
jlach@virginia.edu

John A. Stankovic
Department of Computer Science
University of Virginia,
Charlottesville, VA USA 22904
stankovic@virginia.edu

ABSTRACT

Cyber-physical systems (CPS) are usually safety critical, making systems safety a CPS issue. Many efforts have been made in safety verification of CPS and some effort has been made in safety-guided design of specific CPS, but fewer efforts have been made in a formal science to aid in safety-guided design. One domain crucial to safety-guided design is hazard analysis, which can be challenging for complex dynamic systems like CPS. Recently, systems theoretic process analysis (STPA) has emerged as a promising hazard analysis technique applicable to CPS; however despite its improvement over traditional techniques, it lacks a solid formal (rigorous) approach making much of its application ad-hoc and open to a lot of the issues with non-rigorous methods. This paper presents a formal framework for the hazard identification step in STPA (STPA Step One). We show that the formal framework handles many of the issues that arise in a non-rigorous approach and makes the results from analysis less ambiguous and more complete. We also find that an explicit notion of system components is not necessary for undertaking hazard analysis on the system level much in line with the way systems are analyzed in other systems theory fields.

Categories and Subject Descriptors

J.2 [**Physical Sciences and Engineering**]: Engineering

General Terms

Theory

Keywords

System Safety, Hazard Identification

ICCPS April 8–11, 2013, Philadelphia, PA, USA.

1. INTRODUCTION

Many cyber-physical systems (CPS) are safety-critical. Well-known examples include automotive vehicles, chemical processing plants, and medical devices, with many examples on the devastating effects of these systems have when they exhibit unsafe behaviors. Systems safety is therefore a cyber-physical systems issue. Addressing systems safety includes designing systems to be safe (what has recently been called safety-guided design) and verifying that designed systems meet certain safety constraints.

Many efforts have been made by the CPS software community [11, 2] and hybrid systems and controls communities [12, 13] in safety verification of CPS. Though verification is an important part of the systems safety design process and presents many challenges, verification techniques require complete designs, and redesigning systems after safety issues have been identified at the later verification stages can be costly. Safety-guided design is therefore at least as equally important as verification. Some work has been done in this space for specific CPS (see for example [5]).

Addressing safety-guided design in general requires identifying system safety requirements and the potential ways they can be violated early in the design process, and designing safety controls to eliminate or mitigate these potential violations. Although safety requirements can be identified early in the design phase, the ways in which such constraints can be violated by the system design is not always obvious since such violations typically arise from complex interactions between system components. Understanding how such violations arise is the domain of hazard analysis, which is the focus of this paper.

Recently, a new hazard analysis technique based on systems theory called systems theoretic process analysis (STPA) [6] has emerged as a promising technique for analyzing large, complex sociotechnical systems (see [4, 10, 9]). A growing body of work shows it to be more effective than traditional hazard analysis techniques [14, 4, 3, 8], which typically focus more on components than systems. Its applicability to the earlier stages of safety-guided design has also been demonstrated [3]. The principles of this technique, which emphasize component interactions and systems dynamics, are applicable to CPS because STPA seeks to address the hazard analysis problem in systems where components cross cyber, physical, and social boundaries. One can argue that CPS is

a subset of the kinds of systems that STPA considers.

Despite the promise of STPA, it has a minimal formal framework associated with it, and still requires a significant amount of manual effort from engineers. In addition to specifying the system and its constraints, engineers must manually anticipate system evolution and check, for every possible system command and every possible context in which that command can be given, whether certain conditions on the command results in a hazard (we give a brief description of the STPA process in Section 2). Fortunately, much of the crucial parts of STPA can be formalized[1], reducing the need for manual input beyond specifying the system and its constraints, and creating an opportunity for automating the hazard analysis process by building tools around this formal framework.

The aim of this paper is to present a formal framework (called FSTPA-I), which addresses the hazard identification aspect of STPA (usually called STPA Step One) since this aspect is most crucial to the rest of the analysis. Our framework captures the essence of STPA, and only requires the engineer to specify the system state variables (and the values they can take), the timing properties of control actions, and the system safety constraints. Also, we concentrate on systems whose state variables only take on a finite discrete set of values since this is usually the case at the earlier stages in design. Even though we assume systems have discrete states, our model assumes that systems evolve in continuous time, which is an important assumption for CPS.

There are some differences between our framework and STPA as it is described in the literature. One important difference is that we have no notion of a controller, though one can easily be added to map hazards to components that produce them. Our framework focuses on system states and commands that change those states, and we have found that the notion of a controller is not necessary for undertaking STPA Step One.

We are only aware of one attempt to formalize STPA Step One [14], though the approach only enumerates a set of potentially hazardous conditions, leaving the decision on what constitutes a hazard to the engineer. Our framework arrives at these conclusions on its own by checking such conditions against the system constraints, reducing input from the engineer to only specifying the system as mentioned previously[2]. This eliminates the ambiguity associated with leaving the interpretation of potentially hazardous conditions to the engineer. We have validated our framework by applying it to many of the examples in the STPA literature (to have a good baseline) and found that it identifies clearly the same hazards as those identified by manual and semi-formal application of STPA. In the interest of space, we only provide results from one example (though we use another simpler example to explain parts of our framework).

The main contributions of this paper are:

- Precise definitions of the constructs necessary for specifying a system (including timing properties of its commands) and its constraints for the purposes of hazard analysis.
- Precise definitions of system behavior constructs necessary for STPA analysis.
- Theory on when system behavior is correct and safe with respect to its specification, and when system behaviors represent hazards.
- Precise semantics for STPA hazard analysis concepts.
- General insights about hazard analysis on the system level from formalizing STPA.
- Results from applying our formal framework on a system previously analyzed using a manual approach to STPA, highlighting the advantages of our framework over the manual approach.

[1]Formalizing STPA is not a trivial task as we experienced by undertaking the effort ourselves.

[2]Unfortunately, deciding on system safety constraints is still largely a manual effort, though many practical guidelines in many industries are available for doing so.

2. STPA

We briefly describe the steps involved in the first part of the STPA methodology in order to provide the necessary background for following the rest of the paper and to put our work in perspective. The details provided here are slightly different from what is presented in the STPA literature, but captures the same ideas.

STPA is based on the System Theoretic Accident Model and Process (STAMP) accident model [7]. In this model the system is considered as a control loop (what is termed the safety control structure) and accidents are considered the result of ineffective control. Figure 1 shows a typical (simple) control loop. In complex systems there may be multiple coordinating controllers (and possibly hierarchy in control).

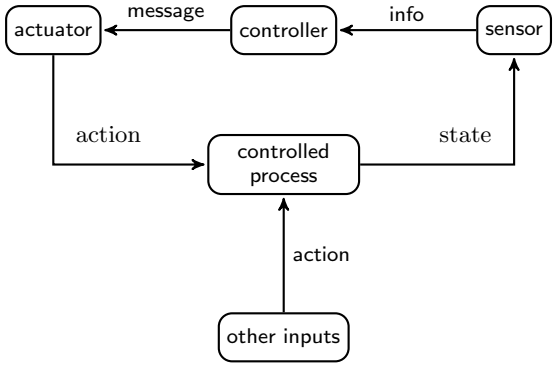

Figure 1: A generic control loop.

According to STAMP, a system exhibits unsafe behavior when the controlled process produces outputs or occupies states that can lead to accidents (*e.g.* a chemical plant begins to release toxic gases into the atmosphere). Hence any behaviors in other components (sensor, controller, or actuator) that cause the controlled process to exhibit unsafe behavior are considered hazardous, and these components must be designed to prevent or mitigate such hazards. The goal of STPA (the hazard analysis technique associated with STAMP) is to identify these potential hazards early in the design stage (and throughout the design and operation of the system) using this system theoretic view of the system.

STPA focuses on what are called control actions (the events that force transitions in the overall system state). It examines each control action under different possible conditions (*e.g.* providing a control action too late) and identifies

whether those conditions lead to hazards and in which system context they lead to hazards. STPA can be broken into two steps: identifying hazards which result from inadequate control (Step One); and determining causal factors for the identified hazards (Step Two). Step One focuses only on control actions and the resulting state changes without special regard to how the control actions are generated. It therefore conceptually focuses on the controlled process and the inputs to it as shown in Figure 2.

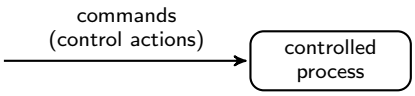

Figure 2: A generic controlled process.

With this view of the system, the hazard analysis essentially proceeds as a 'what-if' analysis on each control action, examining the kinds of behaviors that emerge from the controlled process for each scenario (where the control action is restricted or unconstrained in some way) and determining whether each emergent behavior violates the system safety constraints. The process of STPA Step One is summarized by Figure 3.

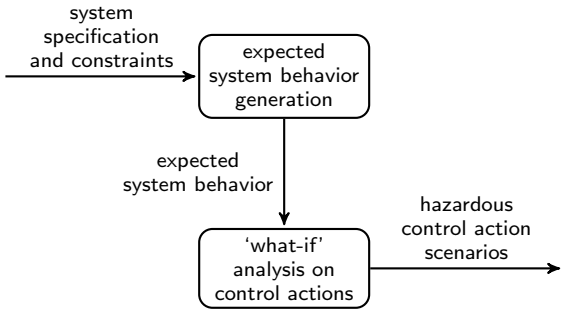

Figure 3: General Steps in STPA One.

3. EXAMPLE SYSTEMS

We describe two example systems which we will use to explain our framework: a simple safety interlock for a high energy power source used by Leveson in [8] to illustrate some aspects of STPA, and the safety of the capture phase for the NASA-JAXA H-II Transport Vehicle (HTV)[1] on which Ishimatsu *et al.* applied STPA [4]. We compare the results from applying our framework on the HTV example later in the Section 5.

3.1 High Energy Power Source

This is a hypothetical high energy power source with a door that opens and closes to allow access to the power source. An operator is responsible for opening and closing the door, and the main hazard is exposure of the operator to the power source. The control components involved are the operator and a computer-based power source control (which turns the power source on and off as appropriate). The safety constraint is that the power source must be off whenever the door is not completely closed (which is the precise definition used here for the door being open).

3.2 NASA H-II Transport Vehicle

The HTV[1] is an unmanned cargo transport vehicle designed by the Japan Aerospace Exploration Agency (JAXA) to transport cargo to the International Space Station (ISS). The authors in [4] state that a principal concern for the HTV is collision with the ISS, which can occur during the berthing phase where the crew must capture the HTV with a robotic arm (SSRMS). Collision could damage both the ISS and HTV and possibly endanger the lives of the ISS crew. Another stated concern is inability to complete the mission resulting from an unintended abort.

The berthing phase of the HTV involves the following steps:

1. HTV reaches berthing point, assumes a grappling position, and maintains distance from ISS (abort can be issued here if any emergency occurs).
2. JAXA ground station enables Flight Releasable Grapple Fixture (FRGF) so HTV can separate (respond to the 'FRGF Separation' command) in case of emergency.
3. ISS crew turns off HTV's autonomous control by sending a 'Free Drift' Command.
4. ISS crew manipulates the SSRMS to grapple (capture) the HTV as quickly as possible (in order to prevent HTV from drifting away).
5. JAXA ground station disables the FRGF so the HTV cannot respond to the FRGF separate command (this prevents unintended separation).

The setup for the system (the components and messages between them) is shown in Figure 4. The ISS crew can control the HTV using a hardware command panel (HCP). JAXA commands are relayed through the ISS. At any point, the ISS crew can send an 'Abort/ Retreat/Hold' command which will cause the HTV to resume autonomous control and execute the required abort sequence. This command is only necessary before capture. An unintended abort is an abort before the HTV can be captured, and this is considered a mission hazard especially if the mission cannot complete because of it.

A collision hazard can occur if the HTV becomes a free-flying object and drifts in the direction of the ISS (or is knocked in the direction of the ISS by the crew making a mistake in grappling the HTV). Also, the FRGF can be separated from the HTV before capture causing it to also become a free-flying object with the potential for collision with the HTV. In addition, whenever the HTV is in an autonomous mode, outside forces on it can be interpreted as disturbances causing it to respond using attitude control. If this occurs when the HTV is captured, it could damage the ISS. All these events should not occur.

4. FSTPA-I

STPA is essentially concerned with system behaviors and how they can result in hazards. Most system behaviors can be described using finite-automata like constructs, and our model of a system behavior follows in this light, but with a few extensions and modifications. Also, our focus is more on the static properties of such a construct as opposed to its output from execution.

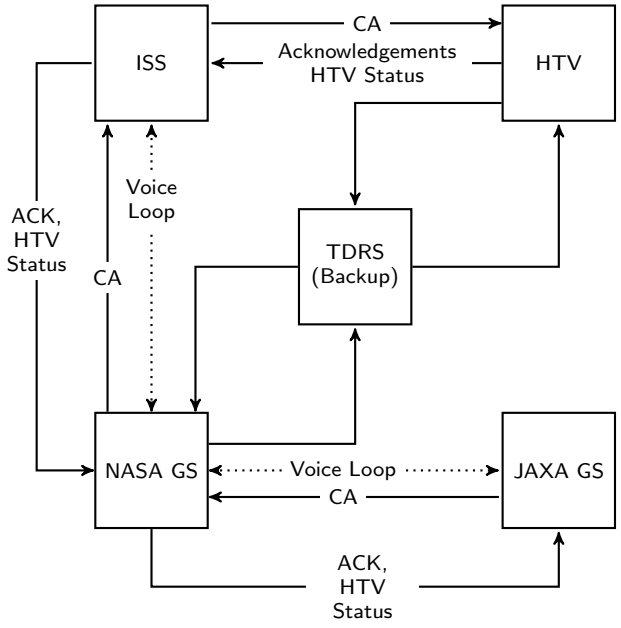

Figure 4: HTV system setup, reproduced according to Figure 6 in [4]. HTV is is the H-II Transport Vehicle. ISS is the Internation Space Station. NASA GS is the NASA Ground Station. JAXA GS is the JAXA Ground Station. TDRS is the Tracking and Data Relay Satellite, which is used as a communication backup. CA is the set of commands that can be sent. The commands sent on all three CA channels are FRGF Sep ENA/INH, Abort/Retreat/Hold, FRGF Separation. The CA channel from the ISS to HTV also includes Free Drift. ACK stands for Acknowledgment.

4.1 Model and Specification Semantics

Our framework is based upon the concept of what we call a system behavior specification, a finite-automaton-like structure that describes the states and kinds of transitions that the system can exhibit given certain constraints. This structure can also be viewed as a directed graph, and we usually take this view in our analysis. Here we describe our concept of state and system state transitions (as well as how they are initiated).

System State

Our concept of state is the same as the concept of state used in state space representations of systems: a system has a number of state variables each of which can take on a set of values (in our case a finite set) and the system state at any point is given by the the collection of values of its state variables. We represent a state as $x = (x_1, \ldots, x_n)$, where x_i is a system variable. The set of values that a system variable can take is $\mathcal{X}_i$. In the power source example, a state can be that the door is closed and the power is on since the state of the door and power source are the two state variables relevant to hazards. If we use the symbol d to represent the door is open, $!d$ to represent the door is closed, p to represent the power is on, and $!p$ to present the power is off, then the example state would be $(!d, p)$ (we use these symbols in the rest of the paper for the power source example).

Control Actions

Each system variable can be caused to take on a particular value from its set of possible values through a control action (causing a change in the system state). In our framework, control actions are specific to system variables and for each value that a system variable can take, there is a control action that causes it to take on that value. Control actions can be *discrete*, meaning they take effect the moment they are issued, or *timed* which means they take some time before they take effect.

Timed control actions have the potential not to complete (*i.e.* they can be stopped before they take effect). This is because timed control actions have a start and stopping point (where they are considered to take effect). Once started, there is no guarantee that the stopping point will be reached. In the power source example, the operator can begin closing the door and stop half-way through before the door is completely closed.

Formally, a control action as a 3-tuple $u = (x_i, v_{x_i}, \rho_t)$ where x_i is the system variable it affects, $v_{x_i} \in \mathcal{X}_i$ is the value that it causes the system variable to take, and $\rho_t \in \{D, T\}$ is the timing property of the control action (D is discrete, T is timed). In a less abstract view of a system, control actions are issued by internal system components or external inputs and disturbances from the environment. We make no distinction between these sources of control actions.

Events

State transitions occur through events. An event in our framework is essentially a finite automaton transition with the added property of an *issue window* which is a time interval within which a control action can be issued and complete. We write an event as a 4-tuple $(x, u, t_{\min}, t_{\max})$, where x is the state or conditions at the beginning of the event (initial state), u is the control action issued to intiate the event, $t_{\min}$ is the minimum time after entering x that elapses before u is issued, and $t_{\max}$ is the maximum time by which u the event completes after entering x with $t_{\min} < t_{\max}$, $t_{\min} \geq 0$, and $t_{\max} > 0$.

One can interpret the event structure as, u can be issued and complete anytime between $t_{\min}$ and $t_{\max}$ after entering x. Having this time property associated with events allows us to deal with time constraints imposed by safety concerns or by the physical nature of the system. In the power source example, the choice of a computer-based controller makes it impossible for the power to be turned off immediately the door is open because the controller must first sense that the door is open. This imposes a time constraint on how fast the controller must respond and a possible event representing this constrained behavior would be $((d, p), !p, 0^+, t_s)$ which means the power must be turned off by t_s after entering the state (d, p), where the door is open and the power source is turned on.

The final (resulting) state entered after the control action completes is the state formed by changing the variable in x affected by the control action u to the value specified by the control action. The resulting state therefore need not explicitly be specified in the event structure. However, one can say that an event δ produces a state x', which we write as $\delta \rightarrow x'$.

We restrict our system model to allow one transition at a time. However, we do allow a concept of concurrent events with the final state determined by applying the control ac-

tions in the system in the temporal order in which the concurrent events are expected to complete. This is because we assume that as long as an event has not completed, it is still in the initial state. However, if another event is initiated from this initial state and completes before the on-going event does, then the new initial state for the on-going event is the final state of the completed event. For example, if δ_1 is currently taking place with initial state $x(\delta_1)$, and two other events (δ_2 and δ_3) are initiated, and if the completion order is δ_1,δ_2,δ_3, then the final state is given by applying u_1 from δ_1 to $x(\delta_1)$ to get $x(\delta_1)'$ (the resulting state), then applying u_2 from δ_2 to the resulting state to get the next state, and finally applying u_3 from δ_3.

System Specification

Most system designs start with a specification of the system and not necessarily a complete description of its behavior. Usually, from this specification, the expected behavior of the system can be generated and proposed designs can be compared against this expected behavior during analysis. In our case, the goal is not to compare proposed system realizations to the expected behavior, but rather to check whether deviations from the expected behavior result in hazards. Nevertheless, the specification must provide enough information to allow the expected behavior to be 'generated' from it. Such information includes the system constraints (safety, logical, or physical), the properties of control actions (whether they are timed or discrete), and initial and final states. A system specification is a 8-tuple $S = (\mathcal{X}, U, C^s_{PL}, C^e_{PL}, H^s, H^e, X_0, X_f)$.

- $\mathcal{X}$ is the set of state variable descriptions $\mathcal{X}_i$ showing the values that each state variable can take.
- X_0 is the set of possible initial states with $|X_0| \geq 1$ since some systems may have multiple initialization points.
- X_f is the set of possible final states with $|X_f| \geq 0$ since some systems may have no final state and continuously evolve (like the power source example) and some systems may have goal states or fail-safe states (like the HTV example which has one goal state and a number of fail-safe states)
- $\mathcal{U}$ is the set of possible control actions. Although these can be inferred from $\mathcal{X}$ one must still specify the timing properties of control actions.
- C^s_{PL} is the set of physically or logically impossible states.
- C^e_{PL} is the set of physically or logically impossible events.
- H^s is the set of state safety constraints (*i.e* a set of states that should not occur because they are hazardous).
- H^e is the set of event safety constraints (*i.e* a set of events that should not occur because they are hazardous). These are useful for specifying timing constraints on conditionally hazardous states (states that should be occupied for a minimum or maximum period of time).

Even though the engineer must specify eight different pieces of information, at the initial stages of design (and at the highest levels of abstraction), the system state space is usually small, making such specifications reasonable to expect.

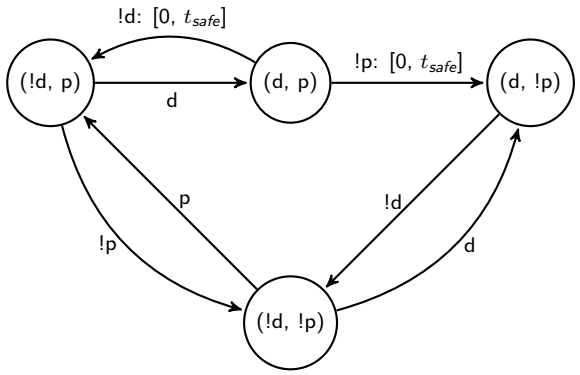

Figure 5: Power source example behavior diagram. Transitions without an explicitly specified issue window have an issue window of $[0, \infty]$

System Behavior Specification

In order to analyze the system, we need a description of its potential behavior given certain constraints. We call this description the system behavior specification. This description must tell us what are the possible states, events, starting states and end states for the particular behavior we wish to examine. A behavior specification as a 4-tuple $\beta_s = (X, X_0, X_f, \Delta)$ where X is the set of possible states, X_0 is the set of possible initial states ($|X_0| \geq 1$), X_f is the set of possible final states ($|X_f| \geq 0$), and Δ is the set of possible events. This description can be visualized as state diagram. Figure 5 shows such a diagram for the power source example.

We must note here that this behavior specification bears similarity to many of the hybrid systems formalisms typically used in system verification. In fact, our behavior specification can have equivalent representations in these formalisms. Our specification is, however, more compact and more amenable to the kind of analysis that STPA One requires (though, again such analysis, with some work, could be converted to analysis using these formalisms).

System Behavior Specification Generation

Since our analysis is on the behavior specification, but our initial input is the system specification, we must derive the the behavior specification from this input. The general steps for this process are as follows:

1. Set X_0 and X_f in β_s to X_0 and X_f from S respectively.
2. Set X initially to all possible states. Remove all states $x : x \in C^s_{PL}$ or $x \in H^s$.
3. Add all possible events given current states in X to Δ. This can be done because states differ by a change in one variable.
4. For all events that match an event in C^e_{PL} or H^e (*i.e* the initial states and control actions are the same and the issue windows overlap) set the issue window to an interval outside the interval specified by the constraint. For example if the constraint is (x, u, t_{min}, ∞) then set the event to $(x, u, 0, t^-_{\min})$.

The resulting behavior specification is correct with respect to the system specification because step 2 ensures that none

of the impossible or hazardous states exist in the behavior specification, and step 4 ensures that timing constraints are not violated. If the system is specified correctly, the resulting behavior specification should be complete (*i.e.* one that has no isolated or unreachable states). Incomplete behavior specifications point to problems in the system specification.

A behavior specification generated from the system specification corresponds to the expected system behavior. Other behavior specifications are possible, and can be obtained by transforming the behavior specification for the expected system behavior. These transformations are what are applied during the hazard analysis stage to identify hazards.

Figure 6 shows a visualization of the generated system behavior for the HTV (in compact form). We do not give the system or behavior specification here (in the interest of space), but we explain the state variables and control actions and the system behavior to show that it is correct and safe (with respect to the specification given in prose when describing the example in Section 3.2):

- *Abort*
  a indicates that the system is in abort state.
  $!a$ means it is not in an abort state. We consider the start state to be an abort state since this is where the mission begins and will reset to if it is aborted and needs to be restarted.
- *Ready*
  r indicates that the HTV is in a position ready for capture (*i.*e. has completed approach towards the ISS).
  $!r$ indicates that the HTV is not ready.
- *FRGF Enable*
  f indicates the FRGF separation is enabled.
  $!f$ indicates that it is disabled.
- *Free Drift*
  d indicates that the HTV is in 'free drift' mode.
  $!d$ indicates that it is in autonomous mode.
- *FRGF Separation*
  s indicates that the FRGF has been separated from the HTV.
  $!s$ indicates it is still connected to the HTV.
- *Capture*
  c indicates that the HTV has been correctly captured.
  ic indicates that the HTV has been incorrectly captured (grappled improperly by the robotic arm).
  $!c$ indicates that it has not been captured.
- *Emergency*
  e indicates an emergency which happens while the HTV is in autonomous mode.
  $!e$ indicates no emergency.

For each state variable value, there is a corresponding control action that causes the variable to take on that value. The only timed control action is the capture action c since the crew must manipulate the robotic arm to capture the HTV which takes some time.

The intended goal state for the HTV is when it is captured and the FRGF separation is disabled given by $(!a, r, !f, d, !s, c, !e)$. The path to this state is the process described in Section 3.2. The other goal states represent the fail safe behavior of the HTV. If an emergency occurs when the HTV is in autonomous mode, it must abort. If incorrect capture occurs, the HTV must separate from the the FRGF and abort. If an unintended separation occurs post capture, the HTV must resume autonomous mode to avoid collision. And if an abort command is sent while the HTV is captured, it must separate (to avoid starting abort sequence while captured) and resume autonomous mode to complete the abort sequence. All other states and events not shown are a violation of the constraints.

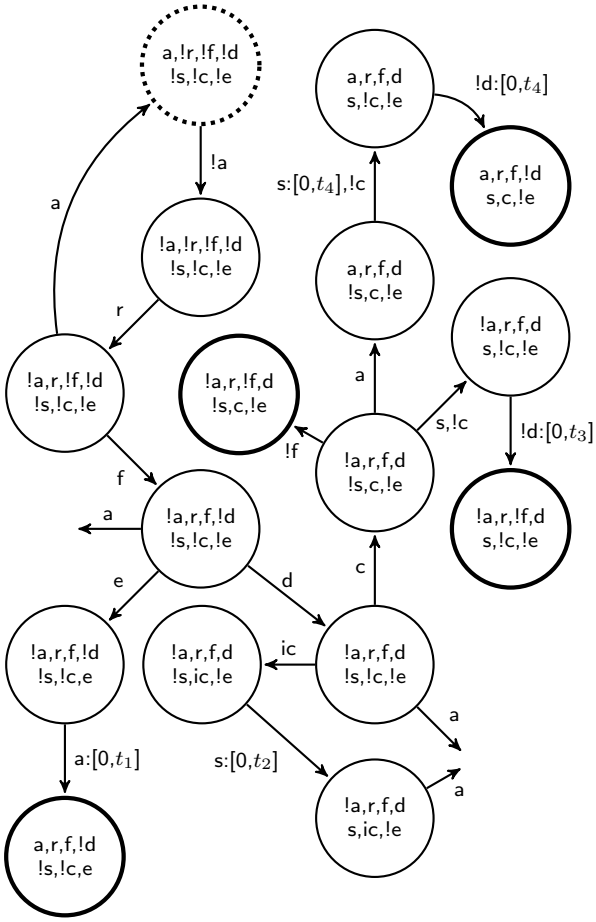

Figure 6: HTV behavior diagram. Transitions without a specified issue window have an issue window of $[0, \infty]$. The state with the dotted outline is the start state and the states with the thicker outlines are the goal states. Abort events without a resulting state or directly connected to the start state indicate an event sequence that eventual ends up in the start state (*i.e* a safe abort of the mission). Transitions with control actions $(s, !c)$ mean $!c$ is issued immediately after the event initiated by s completes.

4.2 Hazards Analysis Semantics

As mentioned previously, STPA proceeds as a 'what-if' analysis on the expected safe behavior of the system. For each control action, four conditions must be examined:

1. The control action is not issued[3] or not followed (*i.e.* the action does not end up affecting system state).

[3]The word 'provided' is used in the STPA literature.

2. The control action is issued when it is unsafe to do so.

3. The control action is issued too early or too late (*i.e* out of sequence).

4. The control action is applied too long or stopped too soon.

To make these conditions more precise, we identified six possible cases:

1. *Omission*: the control action is never issued (or delayed infinitely) in any safe state where it can be issued.

2. *Commission*: the control action is always issued in all safe states where it can be issued.

3. *Premature Initiation*: the control action is issued before it is actually safe to do so (*i.e.* it is issued and completes out of sequence before another control action).

4. *Delayed Initiation*: the control action under consideration is issued too late (*i.e.* it is issued and completes out of sequence after another control action).

5. *Premature Termination*: a timed control action does not complete (*i.e.* the control action is 'stopped' before it can take full effect).

6. *Delayed Termination*: a timed control action stopped too late (*i.e.* it is continued to be applied long after it is safe to do so before stopping).

In our framework, we check these cases by what we call control action conditions (CACs). These are functions of the form $\zeta(u, \beta, S) \rightarrow \beta'$ whose inputs are the control action u, original behavior specification β, and the system specification S and which produce a new behavior specification β' that represents the system behavior given that condition. We have found that only omission and commission need to be formally defined since the other four conditions can be derived from these (as we explain below).

Omission

The omission function, ω, creates a CAC by transforming all events $\delta \in \Delta$ in the original behavior specification β that are initiated by the control action under consideration u to events that never occur (by setting their issue windows to $[\infty, \infty]$, and allows all other physically and logically possible events that can be initiated from the initial states of these events to occur.

Commission

The commission function κ creates a CAC by adding events that can be initiated from states $x \in X$ in β by u, as well as the resulting states from those events, to the β to get β'. Commission essentially allows the control action under consideration to be issued from any safe state where it can be issued.

Premature Initiation

Premature initiation α_ε happens when a control action is initiated and completes out of sequence, before some other control action. This is equivalent to commission where the action is allowed to be issued from any state where it can be issued. Hence commission is equivalent to premature initiation ($\kappa \Rightarrow \alpha_\varepsilon$).

Delayed Initiaion

Delayed initiation α_δ happens when a control action is initiated and completes out of sequence after some other control action. This creates two possibilities: other control actions are initiated before they should (making them unsafe); the delayed control action is initiated when it should not be. The first case is the condition caused by omission, and the second case is the condition caused by commission. A system behavior that represents delayed initiation can therefore be obtained by first applying the omission control action condition and then applying the commission control action condition (*i.e* delayed initiation is $\omega(u, \kappa(u, \beta, S), S)$).

Premature Termination

Premature termination Ω_ε causes events that are initiated by a timed control action to not occur (*i.e* even though the event is initiated, it does not take effect and hence the system never enters the intended resulting state). The definition of omission captures this scenario ($\omega \Rightarrow \Omega_\varepsilon$).

Delayed Termination

Delayed termination Ω_δ causes a timed control action to persist longer than it should, creating two possible conditions: it can allow other potentially hazardous actions to be initiated and complete while still in progress; it completes after a safe transition (event) causing it to take an effect on that state that it should not have. This is equivalent to the delayed initiation condition ($\alpha_\delta \Rightarrow \Omega_\delta$).

Figure 7 shows a visualization of the power source behavior with omission applied on the power on control action.

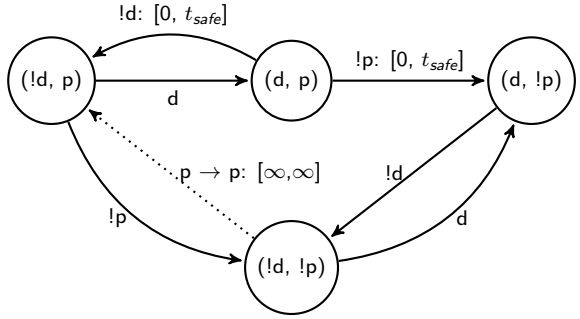

Figure 7: Power source behavior diagram with omission condition on the power on command (p). The dotted line indicates a change to the original behavior specification. Transitions without an explicitly specified issue window have a time window of $[0, \infty]$

With the transformed behavior specifications from the CAC functions, hazard analysis proceeds easily. If any state or event in the new (transformed) behavior specification matches a hazardous state or event constraint in system specification, then the CAC on that control action is hazardous and hazards it produces are those that match the constraints (*i.e.* matching a constraint implies a violation of that constraint).

Table 1: Original STPA hazards analysis results reproduced according to Table 3 in [4]. Event/Command is the event or command related to the hazard. Ineffective control action is either the condition under which the event or command produces a hazard or the command that is provided instead of the one under consideration that results in the hazard.

ID	Event/Command	Ineffective Control Action	Description
1a	FRGF Sep ENA	Not provided Incorrect Out of Sequence	The HTV might not be separated immediately in the emergency situation of the HTV being grappled incorrectly and rotating to collide with the robotic arm.
	FRGF Sep INH	Out of Sequence	
1b	FRGF Sep ENA	Free Drift	The HTV will drift out of the capture box. In combination with no activation command or a late one, the HTV will remain a free-flying object that could collide with the ISS.
	Free Drift	Too Early	
	Capture	Not Provided Incorrect Out of Sequence	
2a	Free Drift	Not provided Incorrect Out of Sequence	The capture will be regarded as a disturbance to the HTV that could trigger an unintended attitude control or even Abort.
	Capture	Out of Sequence	
2b	Free Drift	FRGF Sep	FRGF will be separated from the HTV and become a free-flying object, which is a threat of collision. The HTV will be no longer captured and the mission will end up incomplete.
	Capture	FRGF Sep	
Ca	Capture	Incorrect	The robotic arm could hit the HTV and make it rotate and collide with the ISS.
Cb	Capture	Stopped Too Soon	The HTV is not fixed to the SSRMS and could rotate (windmill) to collide with the arm.
3a	FRGF Sep INH	Not Provided Abort/Retreat/Hold FRGF Sep	In combination with no or late activation command, the HTV will remain a free-flying object that could collide with the ISS.
3b	FRGF Sep INH	Abort/Retreat/Hold	The HTV will make some thrust with remaining captured by the SSRMS. A tension from the arm could be regarded as a disturbance to the hTV that could trigger unintended attitude control.

5. EVALUATION

In this section, we compare the original HTV STPA results from [4] to the results for the same system from our framework.

Table 1 shows the original HTV STPA results for the berthing phase and is reproduced verbatim from the Table 3 in [4]. Each hazard has a particular ID, followed by the cause of the hazard and the description of the hazardous situation for that particular ID. In general we can identify the following hazards for the HTV based on the information provided in [1, 4] (we have mapped the hazards from Table 3 in [4] to these in parenthesis):

H1 The HTV is unable to separate during an emergency in while the crew is trying to capture it (1a).

H2 The HTV becomes a free flying object that can collide with the ISS (1b, 3a, Ca, Cb).

H3 The FRGF becomes a free flying object that can collide with the ISS (2b).

H4 The HTV is autonomous while in contact with the robotic arm and initiates attitude control that damages the robotic arm (2a, 3b).

H5 The mission is unable to restart after an unintended abort (2b).

Table 2 shows the hazards identified by our framework which match the hazards from the original analysis. In the interest of space, we do not present the detailed results of the analysis using our framework; however, starting from the behavior specification we provided in Figure 6 and following the steps described for FSTPA-I, one should be able to produce the appropriate results and show that they match the hazards as given below.

Our framework explicitly accounts for all the possible hazards (H1 to H5) because it considers all possible relevant scenarios. One can also see clearly that the same condition on a control action can result in different hazards (in different contexts). The original STPA also accounts for system hazards H1 to H5, even though it leaves out some scenarios

Table 2: Comparison of Original Results to FSTPA-I results. The meaning each control action symbol is given in the formal description of the HTV berthing process (in Section 4.1). Also, the meaning of each control action condition symbol is given in the section on FSTPA-I Hazard Analysis (Section 4.2). For conciseness, we grouped control actions that have the same control action condition that produces the same hazard as $f_{CAC}(c_1, c_2, \ldots)$. Also, if two or more control action conditions share the same control action for the same hazard, we group them as $f_{CAC_1}, \ldots, f_{CAC_n}(c_1, c_2, \ldots)$.

ID	FSTPA-I Control Action Condition
1a	$\omega, \alpha_\delta(f, s)$; $\alpha_\varepsilon, \kappa(d, c, e)$
1b	$\omega, \alpha_\delta(c, !f)$; $\Omega_\delta(c)$
2a	$\omega, \alpha_\delta(f)$; $\alpha_\varepsilon, \kappa(c)$
2b	$\alpha_\varepsilon, \kappa(s)$
Ca	$\alpha_\varepsilon, \kappa(ic)$; $\omega, \alpha_\delta(f, s)$
Cb	$\Omega_\delta(c)$; $\omega, \alpha_\delta(f, s)$
3a	$\alpha_\varepsilon(s)$; $\omega, \alpha_\delta(!f, a, !d)$
3b	$\alpha_\varepsilon(a, !f)$

that contribute to these hazards. For example, the original analysis does not consider the possibility of the FRGF separation command not being executed even though FRGF was enabled. This could result in 1a (H1).

We found that some of the hazards were confused in the original analysis. One stated cause of hazard 1b is that the *free drift* command is provided instead of the *FRGF enable* command. This should result in hazard 1a not 1b because this only means that capture is started with the FRGF being enabled (H4). The situation described only occurs if capture is not completed quickly after the *free drift* command is given and hence is not an issue necessarily with the *free drift* command. The confusion here is created by the fact that the engineers have switched contexts and not explicitly defined the context. This scenario only results in 1b if the crew fails to capture the HTV after setting it to *free drift*. Our framework captures this whereas the original analysis confuses it.

We also believe that the concepts of "too early" "too late" and "out of sequence" as used in the original analysis are imprecise (and incorrect) and hence lead to confusion in the manual analysis. Our framework defines "out of sequence" using "too early" and "too late": a command can only be "too early" if it happens before it is supposed to and this should therefore be in relation to another event or command; the same goes for "too late" (if it happens after it is supposed to). The relative event could be either a control action casing a state change or time elapsing for events that have time constraints associated with them since are framework provides a notion of time for events.

Without these precise definitions (semantics), these concepts mislead the engineer in the analysis. For example, the original analysis states "If [FRGF enable] is provided too late, it will only delay the capture process". This is not true given the meaning of "too late". This situation should result in no hazard only if it produces a scenario where the command is given after (out of sequence with) the *free drift* command but before capture begins. It however results in hazard 1a if the command is given after capture has began because it creates the potential for the HTV not to separate in case of emergency during capture.

Another issue with the original analysis is that there are quite a number of context-based judgements that are not clear from looking at the original table (see Table 2 in [4]). For example, the authors state that hazard 1b (H2) "If Deactivation is provided too early and capture is not started immediately enough". Here the assumption is that the FRGF has already been enabled. If the FRGF has already been enabled, then deactivation (sending the HTV into *free drift*) can technically not be too early. What is really the case is that capture happens too late. If the *free drift* is indeed provided too early, then it creates the situation discussed above where it is provided before the FRGF is enabled, which results in hazard 1a (H1) not 1b (H2).

6. DISCUSSION

From the comparison above, one can see that our framework makes various factors crucial to the hazard analysis in STPA explicit. Two such factors are timing and order. As pointed out in the previous section, timing and order can be considered incorrectly as is the case with assigning meaning to "out of sequence" using "too early" and "too late". Our framework provides precise semantics for describing these timing related conditions through the use of the two types of control actions (timed and discrete) as well as the definition of control action conditions that capture both issues.

Our framework also provides two interesting insights into hazards analysis on the system level. First, we have found that it is not necessary to have an explicit notion of components that generate the particular system behavior. The extreme view of the system that we take forces an engineer to think more carefully about the system (less about the components) and its potential behaviors at the system level. Of course, some knowledge about the meaning of the system state variables is required, and some of this meaning comes from having an idea of the components. However, our framework focuses attention to the system level, and this can be most useful in the earlier stages of design where component choices may not necessarily have been made.

Second, we have found that context needs to be accounted for more explicitly and completely, especially in the case of conditionally hazardous states. Certain states are only safe in a system if occupied for a particular period of time before transitioning. If this timing constraint is violated, then the system can exhibit hazardous behaviors. Capturing and presenting this precisely can be difficult in a non-rigorous approach and can end up with too conservative constraints being generated, or missing such issues altogether. Also, the same condition on a control action can result in a hazard or not depending on context.

Thomas captures the context issue as well in his work [14]. The difference between Thomas' approach and ours is that his framework enumerates all the possible contexts within which a command can be given and leaves the decision of what results in a hazard to the engineer. This can be a daunting task. The HTV example has 192 possible states and 14 possible transitions from each state accord-

ing to system specification. Even though not all states and state-transition pairs are possible, the possible situations are still large, and larger, more complex systems will have more of such scenarios to evaluate. Our framework examines only the relevant possibilities and identifies hazards using knowledge of safety constraints specified for the system.

7. CONCLUSION

We presented a rigorous approach for identifying potential hazards in CPS using principles from STPA One, the hazard identification step in STPA, a promising hazard analysis technique. We have shown that the formal framework eliminates the issues that arise with applying STPA One in a non-rigorous manner. Our current work in-progress involves applying this framework to more examples in order to refine it if necessary. For the examples we have already considered, the framework works well and helps with explaining scenarios that lead to hazards. Future work includes building tools around this formal framework to help automate the hazard analysis process as well as extending our formal framework to include the second step in STPA (causal factors analysis) where components are considered in order to provide a complete formal foundation for STPA.

8. ACKNOWLEDGMENTS

This work was funded, in part, by NSF Grants EECS-0901686, EECS-1035303, and CNS-1240454.

9. REFERENCES

[1] HTV-1 mission press kit. Rev.A. Technical report, Japan Aerospace Exploration Agency, 2009.

[2] A. Banerjee, K. Venkatasubramanian, T. Mukherjee, and S. Gupta. Ensuring safety, security, and sustainability of mission-critical cyber-physical systems. *Proceedings of the IEEE*, 100(1):283 –299, Jan. 2012.

[3] C. Fleming, M. Spencer, N. Leveson, , and C. Wilkinson. Safety assurance in NextGen. Technical Report NASA/CR-2012-217553, NASA, 2012.

[4] T. Ishimatsu, N. Leveson, J. Thomas, M. Katahira, Y. Miyamoto, and H. Nakao. Modeling and hazard analysis using STPA. In *Conference of the International Association for the Advancement of Space Safety*, 2010.

[5] B. Kim, A. Ayoub, O. Sokolsky, I. Lee, P. Jones, Y. Zhang, and R. Jetley. Safety-assured development of the gpca infusion pump software. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pages 155 –164, Oct. 2011.

[6] N. Leveson. A new approach to hazard analysis for complex systems. In *Int. Conference of the System Safety Society*, August 2003.

[7] N. Leveson. A new accident model for engineering safer systems. *Safety Science*, 42(4):237 – 270, Apr. 2004.

[8] N. Leveson. *Engineering A Safer World: Systems Thinking Applied to Safety*. MIT Press, Cambridge, 2011.

[9] N. Leveson, M. Couturier, J. Thomas, M. Dierks, D. Wierz, B. Psaty, and S. Finkelstein. Applying system engineering to pharmaceutical safety. *Journal of Healthcare Engineering (to appear)*. [Online] http://sunnyday.mit.edu/papers/healthcare-eng-final.doc.

[10] B. D. Owens, M. S. Herring, N. Dulac, N. Leveson, M. Ingham, , and K. A. Weiss. Application of a safety-driven design methodology to an outer planet exploration mission. In *IEEE Aerospace Conference*, 2008.

[11] M. Pajic, R. Mangharam, O. Sokolsky, J. M. G. David Arney, and I. Lee. Model-driven safety analysis of closed-loop medical systems. *IEEE Transactions of Industrial Informatics (TII) (to appear)*, 2013. [Online] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6341078.

[12] S. Prajna, A. Jadbabaie, and G. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *Automatic Control, IEEE Transactions on*, 52(8):1415 –1428, Aug 2007.

[13] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embed. Comput. Syst.*, 6(1), Feb. 2007.

[14] J. Thomas. Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis. Technical Report SAND2012-4080, Sandia National Laboratories, 2012.