Generating Formal Model-Based Safety Requirements for Complex, Software- and Human-Intensive Systems

John Thomas and Nancy Leveson

Complex Systems Research Laboratory, MIT

Cambridge, MA USA

Abstract Systems Theoretic Process Analysis (STPA) is a powerful new hazard analysis method designed to go beyond traditional safety techniques – such as Fault Tree Analysis (FTA) – that overlook important causes of accidents like flawed requirements, dysfunctional component interactions, and software errors. While proving to be very effective on real systems, no formal structure has been defined for STPA and its application has been ad hoc with no rigorous procedures or model-based design tools. This paper defines a formal mathematical structure underlying STPA that can be used to rigorously identify potentially hazardous control actions in a system. A method for using these unsafe control actions to generate formal safety-critical, model-based system and software requirements is presented based on the underlying formal structure, as well as a way to detect conflicts between safety and other functional requirements during early development of the system.

1 Introduction

The introduction of new technology, such as computers and software, is changing the types of accidents we see today. The level of complexity in many of our new systems is leading to accidents in which no components failed but instead unsafe interactions among non-failed components lead to the loss. At the same time, traditional hazard analysis techniques assume accidents are caused by component failures or faults (Vesely and Roberts 1987) and oversimplify the role of humans (Dekker 2005, 2006). Attempts have been made to extend these traditional hazard analysis techniques to include software and cognitively complex human errors, but the underlying assumptions remain the same and do not match the fundamental nature of systems we are building today. For example, most software-related accidents can be traced to incomplete or flawed software requirements (Leveson 1995, Lutz 1992); however, traditional hazard analysis methods like Fault Tree Analysis (FTA) emphasize component failures and overlook unsafe requirements. More powerful hazard analysis techniques are needed.

Formal verification techniques have been useful in ensuring that given requirements are satisfied by an implementation, but do not assist in generating the requirements – i.e. they verify that given requirements are implemented correctly but do not validate that the given requirements are sufficient to enforce safe behaviour of the system. Model checking is one such approach used to ensure that specific software properties or safety requirements are met (Clarke et al. 1999). By developing a formal model of the software and specifying the desired requirements as formal logic statements, automated algorithms can be used to check the software model and either verify that the stated requirements are upheld in the assumed environment or provide a counterexample or scenario in which the requirements are violated. However model checking requires a detailed model of the software implementation to be checked, and it does not validate that the requirements to be checked are adequate to enforce safe behaviour from an encompassing system perspective.

Formal methods have also been developed to refine high-level goals and requirements into more precise specifications of software behaviour (Darimont and Lamsweerde 1996, Lamsweerde et al. 1998). However, these methods do not interface with the system hazard analysis outputs. Other work has developed criteria for software requirements completeness (Heimdahl and Leveson 1996, Leveson 2000). This approach provides some basic guidance by identifying common ways in which a requirements specification can be incomplete or inconsistent. However, this effort focuses on desirable criteria for all software requirements in general; additional work is necessary to verify that software requirements are not only complete and consistent, but *safe*. It also does not necessarily identify requirements that are related to the specific application involved, such as when the throttle on an aircraft needs to be advanced to prevent a stall.

While all of these techniques are useful for their intended goals, they do not solve the problem of identifying or generating the safety requirements. This paper presents a method for generating and validating safety-critical requirements using a new hazard analysis method, STPA (System-Theoretic Process Analysis) that is based on a new accident causation model called STAMP (System-Theoretic Accident Model and Processes).

1.1 STAMP and STPA

STAMP is a model of accident causation that treats safety as a control problem rather than as a failure problem (Leveson 2012). While unsafe control includes inadequate handling of failures, it also includes system and software design errors and erroneous human decision making. In STAMP, accidents are viewed as the result of inadequate enforcement of constraints on system behaviour. The reason behind the inadequate enforcement may involve classic component failures, but it may also result from unsafe interactions among components operating as designed

and consistent with their specified requirements – or from erroneous control actions by software or humans.

STAMP is based on the observation that there are four types of hazardous control actions that can lead to accidents:

- A control action required for safety is not provided or is not followed.
- An unsafe control action is provided that leads to a hazard.
- A potentially safe control action is provided too late, too early, or out of sequence.
- A safe control action is stopped too soon or applied too long.

One potential cause of a hazardous control action is an inadequate process model used by human or automated controllers. The process model contains the controller's understanding of:

1. the current state of the controlled process
2. the desired state of the controlled process
3. the ways the process can change state.

It is used by the controller to determine what control actions are needed. In software, this process model is usually implemented in variables and may be embedded in the algorithms used. For humans, the process model is often called the 'mental model'. Software and human errors frequently result from incorrect process models, e.g., the software thinks the spacecraft has landed and shuts off the descent engines. Accidents can therefore occur when an incorrect or incomplete process model causes a controller to provide control actions that are hazardous. While process model flaws are not the only cause of accidents involving software and human errors, they are a major contributor.

STPA is a hazard analysis technique built on STAMP. Identifying the hazardous control actions for the specific system being considered is the first step in STPA. These unsafe control actions can be used to identify basic constraints (requirements) on the behaviour of the controller in order to ensure that unsafe behaviour does not result. Additional analysis can then be performed to identify the detailed scenarios leading to the violation of these safety constraints, potentially identifying the need for even more requirements. As in any hazard analysis, the detailed scenarios are then used to alter the design to eliminate or control the hazards in the system design. Additional design features to control hazards, in turn, may generate new hazards or new paths to hazards and lead to additional safety-critical requirements.

This paper presents a formal technique based on STPA that can be used to identify hazardous control actions and generate formal, model-based specifications that enforce safe behaviour in the system.

1.2 Overview of the STPA process

Before beginning an STPA hazard analysis, potential accidents and corresponding system-level hazards are identified. As an illustrative example, consider a simple automated door control system for a train. The accidents to be considered are: injury to a person caused by falling out of the train, a person is hit by a closing door, or people are trapped inside a train during an emergency. The system-level hazards relevant to this definition of an accident include:

H-1. Doors close on a person in the doorway.

H-2. Doors open when the train is moving or not aligned with a station platform.

H-3. Passengers/staff are unable to exit during an emergency.

STPA is performed on a functional control diagram of the system, shown in Figure 1 for the train door controller. STPA has two main steps.

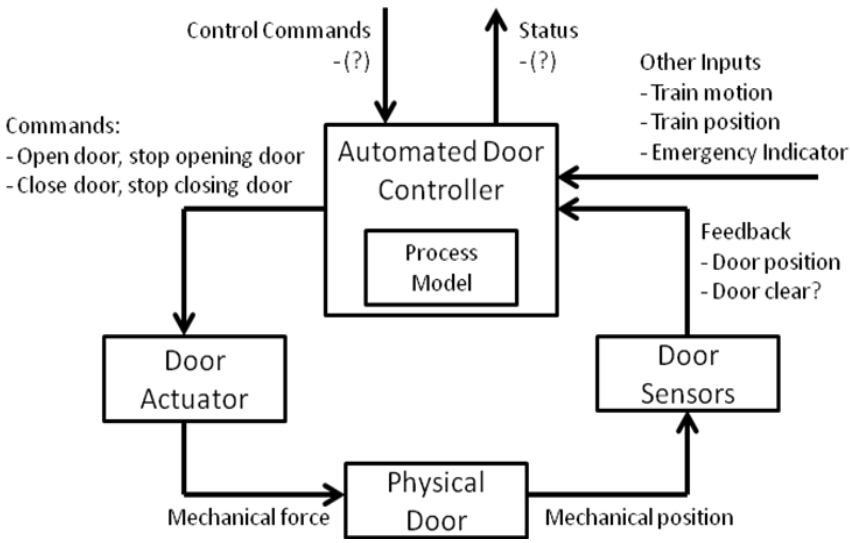

Fig. 1. Simplified control diagram for an automated door controller

STPA Step One. The first step of STPA identifies control actions for each component that can lead to one or more of the defined system hazards. The four general types of hazardous control actions shown above can be used to guide the engineering team as they perform this step. For example, one hazardous control action would be a *close door* command that is issued while a person is in the doorway.

STPA Step Two. The second step of STPA examines each control loop in the safety control structure to identify potential causal factors for each hazardous control action, i.e., the behaviours that can lead to the hazardous control actions iden-

tified in Step One. 0shows a generic control loop that can be used to guide this step. While STPA Step One focused on the provided control actions (the upper left corner of 0), STPA Step Two expands the analysis to consider causal factors along the rest of the control loop.

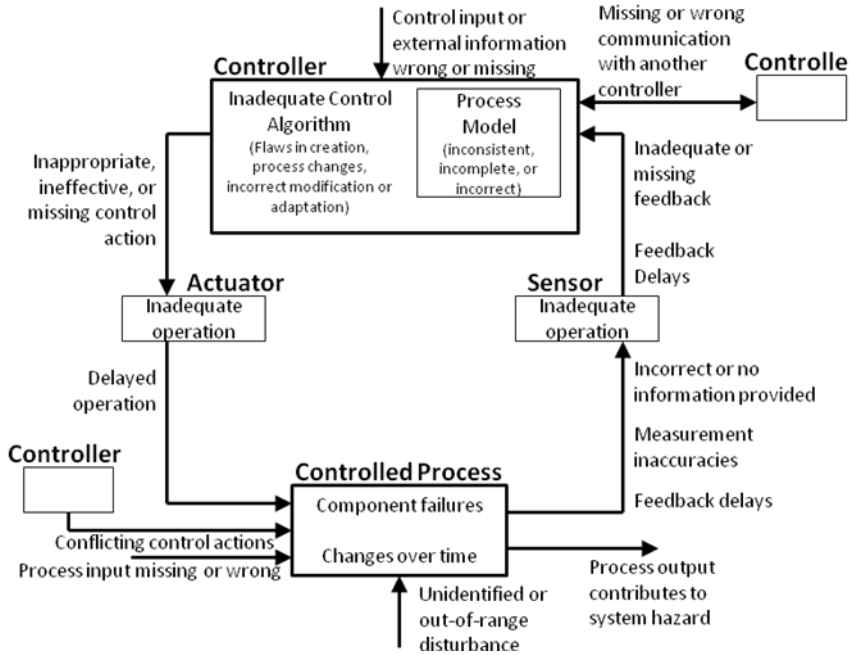

Fig. 2. General control loop with causal factors

Consider the example above where the hazardous control action is to command the doors closed on a person in the doorway. STPA Step Two would identify one potential cause of that action as an incorrect belief that the doorway is clear (an incorrect process model). The incorrect process model, in turn, may be the result of inadequate feedback provided by a failed sensor or the feedback may be delayed or corrupted. Alternatively, the designers may have omitted a feedback signal.

Once the second step of STPA has been applied to determine potential causes for each hazardous control action identified in STPA Step One, the causes should be eliminated or controlled in the design.

STPA has been described in other places (Leveson 2012) and is not described in further detail here due to space limitations. The goal of this paper is to formalize the process and identify tools that can be used to perform it.

2 Formal syntax for hazardous control actions

A hazardous control action in the STAMP accident model is a critical output of STPA Step One and forms the basis for STPA Step Two. In this section a formal syntax is defined for hazardous control actions. Sections 3 and 4 describe how STPA hazard analysis can be partially automated based on this syntax and how model-based safety requirements can be generated.

Hazardous control actions can be expressed formally as a four-tuple (S, T, CA, C) where:

- S is a controller in the system that can issue control actions. The controller may be automated or a human.
- T is the type of control action. There are two possible types: *Provided* describes a control action that is issued by the controller while *Not Provided* describes a control action that is not issued.
- CA is the specific control action or command that is (or is not) output by the controller.
- C is the context in which the control action is (or is not) provided.

For example, in the case of the automated train door controller from Section 1.2, consider the following hazardous control action: the train door controller provides the open door command while the train is moving. This control command can be expressed as (S, T, CA, C) where:

S = Train door controller.
T = Provided.
CA = Open door command.
C = Train is moving.

Each element of a hazardous control action is a member of a larger set, i.e. the following properties must hold:

1. $S \in \mathbb{S}$, where $\mathbb{S}$ is the set of controllers in the system.
2. $T \in \mathcal{T}$, where $\mathcal{T}$ = {Provided, Not Provided}.
3. $CA \in \mathcal{CA}(S)$, where $\mathcal{CA}(S)$ is the set of control actions that can be provided by controller S.
4. $C \in \mathcal{C}(S)$, where $\mathcal{C}(S)$ is the set of potential contexts for controller S.

To assist in enumerating or aggregating individual contexts, the context C can be further decomposed into variables, values, and conditions:

- V is a variable or attribute in the system or environment that may take on two or more values. For example, *train motion* and *train position* are two potential variables for a train.
- VL is a value that can be assumed by a variable. For example, *stopped* is a value that can be assumed by the variable *train motion*.

- CO is a condition expressed as a single variable/value pair. For example, *train motion is stopped* is a condition.
- The context C is the combination of one or more conditions and defines a unique state of the system or environment in which a control action may be given.

The following additional properties related to the context of a hazardous control action can therefore be defined:

5. V ∈ $\boldsymbol{\mathcal{V}}$(S), where $\boldsymbol{\mathcal{V}}$(S) is the set of variables referenced in the system hazards $\boldsymbol{\mathcal{H}}$.
6. VL ∈ $\boldsymbol{\mathcal{VL}}$(V), where $\boldsymbol{\mathcal{VL}}$(V) is the set of values that can be assumed by variable V.
7. CO = (V, VL) ∈ $\boldsymbol{\mathcal{CO}}$(S), where $\boldsymbol{\mathcal{CO}}$(S) is the set of conditions for controller S.
8. C = (CO_1, CO_2, ...), where each CO_i is independent. That is, no two CO_i refer to the same variable V.

Finally, each hazardous control action must be linked to a system-level hazard:

9. To qualify as a hazardous control action, the event (S, T, CA, C) must cause a hazard H ∈ $\boldsymbol{\mathcal{H}}$, where $\boldsymbol{\mathcal{H}}$ is the set of system level hazards.

A hazardous control action expressed as a four-tuple (S, T, CA, C) must satisfy the above properties 1-9.

3 Identifying hazardous control actions

An informal procedure for identifying hazardous control actions has previously been described in (Thomas and Leveson 2011). This section defines a formal method that can be used to automate much of that manual process.

Using the formal definitions in Section 2, a set of potentially hazardous control actions can be enumerated once certain information about the system is known. The information needed is:

$\boldsymbol{\mathcal{H}}$: the set of system-level hazards
Ş: the set of controllers in the system
$\boldsymbol{\mathcal{CA}}$(S): the set of control actions for each controller S
$\boldsymbol{\mathcal{V}}$: the set of variables referenced in the hazards $\boldsymbol{\mathcal{H}}$
$\boldsymbol{\mathcal{VL}}$(V): the set of potential values for each variable V.

Most, if not all, of this information can be determined well in advance of the detailed design of a system. The set $\boldsymbol{\mathcal{H}}$ is typically determined during the Preliminary Hazard Analysis (PHA) of the system. The sets Ş and $\boldsymbol{\mathcal{CA}}$(S) can be extracted from a preliminary control structure of the system. The set $\boldsymbol{\mathcal{V}}$ is identical to the process model variables in the control structure, and can be extracted from the

set of hazards $\mathcal{H}$. The potential values $\mathcal{VL}$(V) are also found in the process model, and can be defined once $\mathcal{V}$ is known.

Given this basic information about the system, properties 1-8 from Section 2 can be applied to automatically generate a list of potential hazardous control actions in the form of combinations of (S, T, CA, C). First, a controller S is selected from the set $\mathbf{Ş}$. Then the set of conditions $\mathcal{CO}$(S) is generated by pairing each variable in $\mathcal{V}$ with each value in $\mathcal{VL}$(V). Then the set of contexts $\mathcal{C}$ is generated by combining each independent condition from $\mathcal{CO}$(S). Finally, the list of potentially hazardous control actions for the selected controller S is generated by combining each element of $\mathcal{T}$, $\mathcal{CA}$(S), and $\mathcal{C}$(S). This process can be repeated for each controller S in the set $\mathbf{Ş}$.

This process generates a set of potential hazardous control actions in which properties 1-8 from Section 2 are guaranteed to be satisfied. Because a detailed behavioural model of the system typically does not exist during the earliest phases of development, it may not be possible to automatically apply property 9. However, this final step can be performed by the engineering team. Because the algorithm above generates combinations that satisfy all other criteria, the generated list is a superset of the actual hazardous control actions. Therefore the remaining part of the task that is not automated is a trimming exercise: the engineering team does not need to propose any new hazardous control actions, they only need to remove non-hazardous control actions from the generated list based on their knowledge of the physics and other engineering properties of the overall system.

For example, in Tables 1 and 2 the engineering team would need to fill in the columns on the far right:

Table 1. Example hazardous control action table for door open command NOT provided

Control action	Train motion	Emergency	Door obstruction	Hazardous?
Door open command NOT provided while…	(doesn't matter)	Yes	(doesn't matter)	Yes (see H-3)
Door open command NOT provided while…	(doesn't matter)	(doesn't matter)	Closing on obstruction	Yes (see H-1)[1]
Door open command NOT provided while…		(all others)		No

For each potential hazardous control action in Table 1 (T = Provided), timing information such as potentially hazardous delays within a given context should also be considered. For example, suppose it is not hazardous to provide a door open command while the train is stopped and there is an emergency. In fact, this behaviour may be exactly what is expected of the system. However, providing the door open command *too late* in that context could certainly be hazardous even if the control action is eventually provided. This condition can be addressed by adding

[1] Of course, the system should be designed so the doors never close on a person. However, in the event that the doors do close on a person, the system must be designed to immediately open the doors (i.e. minimize H-1).

the columns *hazardous if provided too early* and *hazardous if provided too late* as illustrated in the second row of Table 3.

Table 2. Example hazardous control action table for door open command provided

Control action	Train motion	Emergency	Train position	Hazardous?
Door open command provided while…	Moving	(doesn't matter)	(doesn't matter)	Yes (see H-2)
Door open command provided while…	Stopped	Yes	(doesn't matter)	No
Door open command provided while…	Stopped	No	Not at platform	Yes (see H-2)
Door open command provided while…	Stopped	No	At platform	No

Table 3. Example hazardous control action table including timing information

Control action	Train motion	Emergency	Train position	Hazardous?	Hazardous if provided too early?	Hazardous if provided too late?
Door open command provided while…	Moving	(doesn't matter)	(doesn't matter)	Yes (see H-2)	Yes (see H-2)	Yes (see H-2)
Door open command provided while…	Stopped	Yes	(doesn't matter)	No	No	Yes (see H-3)
Door open command provided while…	Stopped	No	Not at platform	Yes (see H-2)	Yes (see H-2)	Yes (see H-2)
Door open command provided while…	Stopped	No	At platform	No	No	No

Once the hazardous control actions have been identified, each action can be examined to define a safety constraint for the system. For example, consider the hazardous control action from the first row of Table 1:

Hazardous control action. Train door controller provides the open door command while the train is moving.

Safety constraint. Train door controller must not provide the open door command while the train is moving.

While this simple example is fairly obvious and would probably not require the use of a formal method, our experience using STPA on real systems such as spacecraft (Ishimatsu et al. 2010), the air transportation system (Fleming et al.

2011, Laracy 2007), and missile defence systems (Pereira et al. 2006) has led to the identification of safety-critical requirements that were never considered during the normal development of these systems.

4 Generating model-based specifications

Because hazardous control actions have been defined with a formal representation, it is possible to compare these actions against an existing formal model-based specification (e.g. SpecTRM-RL) to determine whether the hazardous control actions can occur in an existing design. Furthermore, if no formal specification exists, it is possible to automatically generate the parts of the specification necessary to ensure hazardous behaviour is prevented.

The following functions can be defined from the set of hazardous control actions:

HP(H, S, CA, C). This function is *True* if and only if hazard H results from controller S providing command CA in context C. This function is defined for all $H \in \boldsymbol{\mathcal{H}}$, $S \in Ş$, $CA \in \boldsymbol{\mathcal{CA}}(S)$, $C \in \boldsymbol{\mathcal{C}}(S)$.

HNP(H, S, CA, C). This function is *True* if and only if hazard H results from controller S not providing command CA in context C. This function is defined for all $H \in \boldsymbol{\mathcal{H}}$, $S \in Ş$, $CA \in \boldsymbol{\mathcal{CA}}(S)$, $C \in \boldsymbol{\mathcal{C}}(S)$.

The formal requirement specification or control algorithm to be generated can be expressed as the following function:

R(S, CA, C). This function is *True* if and only if controller S is required to provide command CA in context C. This function must be defined for all $S \in Ş$, $CA \in \boldsymbol{\mathcal{CA}}(S)$, $C \in \boldsymbol{\mathcal{C}}(S)$.

The goal, then, is to compute the function R such that hazardous behaviour is prevented. Namely, any control action that is hazardous in a given context must not be provided by the control algorithm in that context:

$$\forall H \in \boldsymbol{\mathcal{H}}, S \in \mathbf{Ş}, CA \in \boldsymbol{\mathcal{CA}}(S), C \in \boldsymbol{\mathcal{C}}(S): HP(H, S, CA, C) \Rightarrow \neg R(S, CA, C)$$

In addition, if a control action that is absent in a given context will produce a hazard, then the control action must be provided by the control algorithm in that context:

$$\forall H \in \boldsymbol{\mathcal{H}}, S \in \mathbf{Ş}, CA \in \boldsymbol{\mathcal{CA}}(S), C \in \boldsymbol{\mathcal{C}}(S): HNP(H, S, CA, C) \Rightarrow R(S, CA, C)$$

The required behaviour R can then be generated to satisfy these two criteria. Any behaviour appearing in HNP must appear in R, and any behaviour that appears in HP must be absent from R.

The resulting controller requirements (R) can be converted into a formal model-based requirements specification language such as SpecTRM-RL (Leveson

et al. 1999). For example, Figure 3 contains a formal SpecTRM-RL specification for the train door example. The three columns on the right specify three contexts in which the open doors command must be provided: when the train is aligned and stopped, or when the train is stopped and an emergency exists, or when the doors are closing on a person and the train is stopped. The right two columns specify behaviour that is required to prevent the system hazards, and were automatically generated by a software tool that implements the procedure above. The first column specifies behaviour that is necessary for the intended function of the system, not to avoid hazards, and therefore is not automatically generated by the procedure above.

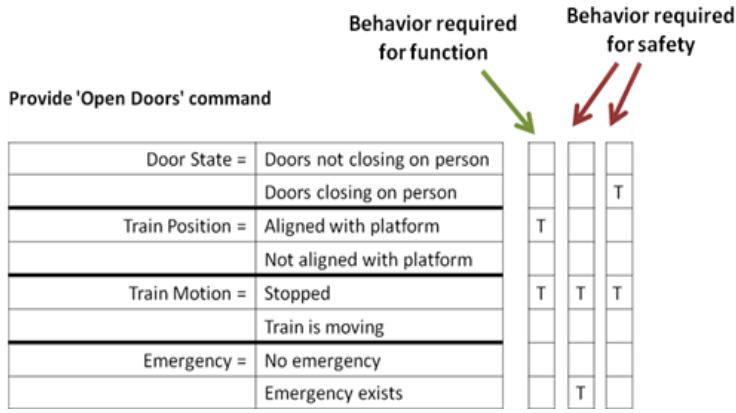

Fig. 3. Generated SpecTRM-RL table for the door open command

4.1 Automated consistency checking

If the same behaviour appears in HNP and HP, then no R can satisfy both criteria. The following additional criterion can be defined to detect these conflicts and ensure that a solution R exists:

$\forall\ H_1 \in \mathcal{H},\ H_2 \in \mathcal{H},\ S \in \mathcal{S},\ CA \in \mathcal{CA}(S),\ C \in \mathcal{C}(S)$: $HP(H_1, S, CA, C) \Rightarrow \neg HNP(H_2, S, CA, C)$

The third criterion above is a consistency check that can be applied to the hazardous control actions even before the formal specification R is generated. If the third criterion does not hold, there is a design or requirements flaw in the system. Both action and inaction by controller S will lead to a hazard and violate a safety constraint. Although the conflict cannot be automatically resolved, it can be automatically detected and flagged for review by the engineering team.

For example, suppose the train from Section 3 is moving and there is an onboard emergency such as a fire. HNP will state that not opening the door in this situation is hazardous due to H-3. However, HP states that opening the door in this

situation is hazardous due to H-1. This is an example of a conflict that can be automatically detected by the formal criterion above.

In general, the system design may need to be reviewed or revised to ensure that conflicts are handled appropriately. Ideally, the conflict would be eliminated by a design change. If elimination is not possible, hazard mitigation or reduction may be performed by placing constraints on other control actions in the system (constraints on HP and HNP for other S and CA). For example, if the train is moving and there is an emergency then a constraint can be defined for the braking system controller such that the brakes are always applied in this situation. The *Train Motion* variable will soon transition from *moving* to *stopped*, thereby resolving the conflict for the train door controller.

If hazard mitigation or reduction is not possible, the conflict could alternatively be handled based on hazard severity or priority.

4.2 Extending hazard analysis to non-safety goals

One of the columns in Figure 3 specifies behaviour that is necessary for the intended function of the system, not to avoid safety hazards. However, this column is clearly important; without it, the whole train system could not achieve its purpose of transporting people.

Although the procedures thus far have focused on safety-related behaviour, the functional behaviour of the system can be defined in the same way and functional specifications can be generated along with the safety-related specifications by following a similar method. More specifically, in addition to HP and HNP – which capture hazardous control actions – a new function FP can be introduced to capture control actions that are needed to achieve functional goals:

FP(F, S, CA, C). This function is *True* if and only if system-level function F must be achieved by controller S providing command CA in context C to achieve a system-level function F.

The function FP can be defined by identifying which control actions in each context are necessary to achieve the system-level functions $\boldsymbol{\mathcal{F}}$. The same process used in Section 3 to identify hazardous control actions can be applied to the system-level functions $\boldsymbol{\mathcal{F}}$ as opposed to the system-level hazards $\boldsymbol{\mathcal{H}}$. The required behaviour R can then be computed as in Section 4, but with an additional criterion to capture the functional behaviour:

$$\forall \mathrm{F} \in \boldsymbol{\mathcal{F}}, \mathrm{S} \in \boldsymbol{\mathcal{S}}, \mathrm{CA} \in \boldsymbol{\mathcal{CA}}(\mathrm{S}), \mathrm{C} \in \boldsymbol{\mathcal{C}}(\mathrm{S}): \mathrm{FP}(\mathrm{F}, \mathrm{S}, \mathrm{CA}, \mathrm{C}) \Rightarrow \mathrm{R}(\mathrm{S}, \mathrm{CA}, \mathrm{C})$$

Applying this criterion, any behaviour appearing in FP must also appear in R. Note that if the same behaviour appears in FP and HP, then there is a design or requirements flaw in the system because the same control action is both necessary to achieve a system-level function and prohibited because it presents a system-level hazard. In that case, no R would exist that prevents the hazards while achiev-

ing the system functions. The following additional criterion can therefore be defined:

$\forall$ H $\in \boldsymbol{\mathcal{H}}$, F $\in \boldsymbol{\mathcal{F}}$, S $\in \boldsymbol{\mathcal{S}}$, CA $\in \boldsymbol{\mathcal{CA}}$(S), C $\in \boldsymbol{\mathcal{C}}$(S): HP(H, S, CA, C) $\Rightarrow \neg$FP(F, S, CA, C)

This final criterion is a consistency check to detect conflicts between hazardous and functional behaviour. As before, these conflicts cannot be automatically resolved, but they can be automatically detected and flagged for review by the engineering team. The full SpecTRM-RL model in Figure 3 was generated automatically by a software tool that implements the five criteria from this section.

5 Conclusions

This paper presents a formal structure underlying STPA hazard analysis and a corresponding method to systematically identify hazardous control actions in a system. A set of formal criteria have been defined to automate much of the hazard analysis process even when a detailed model of the system or software components has not yet been developed. A method for using the STPA hazard analysis results to generate formal safety-critical, model-based system and software requirements is also presented. The generated formal requirements are executable and can be imported into the SpecTRM toolset (Leveson et al. 1999) for simulation if needed.

The ability to formally translate between the hazard analysis and model-based requirements also permits the automatic detection of requirements flaws or conflicts in which not all hazards are prevented. The criteria necessary to detect such conflicts are defined in Section 4 and a prototype tool has been developed to automatically identify such conflicts in a set of formal requirements. The same approach can be applied to generate functional requirements in parallel with the safety requirements. As a result, conflicts between safety and functional goals can also be detected by evaluating the additional criteria defined in Section 4 above.

Acknowledgements This work was partially supported by NASA Contract NNL10AA13C, a JAXA research grant, and a fellowship provided by Sandia National Laboratory.

References

Clarke EM, Grumberg O, Peled D (1999) Model checking. MIT Press

Darimont R, Lamsweerde Av (1996) Formal refinement patterns for goal-driven requirements elaboration. Proc 4th ACM SIGSOFT symposium on Foundations of software engineering, San Francisco, California, United States

Dekker S (2005) Ten questions about human error: a new view of human factors and system safety. Lawrence Erlbaum Associates, Mahwah NJ

Dekker S (2006) The field guide to understanding human error. Ashgate, Aldershot, UK; Burlington, VT

Fleming C, Spencer M, Leveson N, Wilkinson C (2011) Safety assurance in Nextgen. NASA Technical Report

Heimdahl MPE, Leveson NG (1996) Completeness and consistency in hierarchical state-based requirements. IEEE Trans Softw Eng, 22(6):363-377

Ishimatsu T, Leveson N, Thomas J, Katahira M, Miyamoto Y, Nakao H (2010) Modeling and hazard analysis using STPA. Paper presented at the Conference of the International Association for the Advancement of Space Safety, Huntsville, Alabama

Lamsweerde Av, Letier E, Darimont R (1998) Managing conflicts in goal-driven requirements engineering. IEEE Trans Softw Eng 24(11):908-926

Laracy JR (2007) A systems-theoretic security model for large scale, complex systems applied to the US air transportation system. MIT, Engineering Systems Division

Leveson N (1995) SafeWare: system safety and computers. Addison-Wesley, Reading, Mass..

Leveson N (2000) Completeness in formal specification language design for process-control systems. Proc 3rd workshop on Formal methods in software practice, Portland, Oregon, USA

Leveson N (2012) Engineering a safer world: systems thinking applied to safety. MIT Press, Cambridge, Mass.

Leveson NG, Heimdahl MPE, Reese JD (1999) Designing specification languages for process control systems: lessons learned and steps to the future. Paper presented at the Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, Toulouse, France

Lutz RR (1992) Analyzing software requirements errors in safety-critical, embedded systems. Paper presented at the International Conference on Software Requirements

Pereira S, Lee G, Howard J (2006) A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system. Paper presented at the AIAA Missile Sciences Conference, Monterey, CA

Thomas J, Leveson N (2011) Performing hazard analysis on complex, software- and human-intensive systems. International System Safety Conference, Las Vegas, NV

Vesely WE, Roberts NH (1987) Fault tree handbook. US Independent Agencies and Commissions