Using Formal Notations to Augment a Hazard Analysis Method

Fryad M. Rashid, MSc; Clemson University/School of Computing; Clemson, SC, USA

John D. McGregor, PhD; Clemson University/School of Computing; Clemson, SC, USA

Keywords: Error ontology, Hazard Analysis, Safety-Critical Systems

Abstract

Safety-critical systems are those systems whose failure could result in loss of life. For that reason, it needs specific development activities in the software development life cycle to ensure that the system will operate safely. The overall objective of this research is to develop a theoretical framework that identifies unsafe functional behavior for each component in the system. Current safety analysis methods do not use formal methods to find hazardous conditions in a design. We provide guidance concerning mathematical notations to formalize an error ontology used in the architecture descriptions of systems represented in AADL (Architecture Analysis and Design Language) and to improve the rigor of STPA (Systems-Theoretic Process Analysis), a hazard analysis method. In this study, we investigated how a formalized error ontology could assist in identifying unsafe behavior. The ontology can aid in identifying mathematical expressions for each error flow in the canonical feedback control loop architecture. The results of our studies have shown that providing a formal notation for the feedback control loop and providing formal specification for the error ontology lead to finding hazards in the operational system context that other methods miss. By augmenting STPA with an error ontology described in a formal notation, we are able to find more hazards.

Introduction

Development of safety-critical systems requires an explicit design to manage variations, component failures and emergent behavior between system components resulting in hazards. Process control architectures are the core of safety-critical systems used for defining process control variables and computing their change. Based on those variations, the systems can be classified into two types: open-loop and closed-loop. The open-loop system performs a function on the input variable to determine how to control the process. For example, for a lamp, the output of the system is light and the switch is the controller of the lamp. Any change in light has no effect on the on-off position of the switch or its controlling action. But, the closed-loop system measures changes in the output variable and feeds it back as an input variable into the system. Home heating systems and cruise control systems are examples of closed-loop systems (Bergen, 2004).

This paper describes an approach to providing a description for each open-loop component in the canonical feedback control loop architecture based on a functional notation and evaluation theory. Also, we describe an error ontology in the form of algebraic notations to describe a relationship among the component functions and different types of errors. This will allow us to augment the current accepted standard, STPA (System-Theoretic Process Analysis) (Leveson, 2011), with a
theoretical framework which consists of component functions and error event conditions described in a formal notation to identify hazards in the feedback control loop that other methods miss.

The main reason for doing this research is to augment the hazard analysis processes of STPA making the process more comprehensive to uncover additional hazards and to reduce the reliance on human expertise. STPA identifies hazardous control actions based on the values of the process model variables which are used to create safety constraints for the controller. To identify hazards, each relevant value will be examined to determine the context of the control action. This context will be used by the controller to decide which command to send. If one of the existing values in the process model is not compatible or reliable within the control action context, that action can be considered an unsafe action. From an architecture perspective, what will happen if the value/variable is not defined or missed in that context? Does it lead to missing safety constraints for the system? How does STPA solve this problem? For modeling the system, we need to know the specific information about each component in the feedback control loop such as functions, inputs, outputs, variables, and values. In addition, we need to know specifically how the process model in the controller makes a decision based on the existing variables and their values.

Specifically, operation of the safety-critical system can be expressed as a function relating system inputs to outputs. There is a relationship among inputs and outputs where each output value will be related to one or more input values. The system consists of components whose interaction achieves the function of the component. When the system is working correctly, it means that the system satisfies all functional requirements. This, however, does not mean the system is safe. The system should be analyzed regarding the safety aspects and verified according to its safety requirements.

The overall objective of this research is to identify execution flows that produce erroneous values that can lead to the component failing to satisfy its functional requirements. This allows us to identify unsafe functional behavior for each component in the system. We need to identify safety constraints for unsafe functions because any unsafe behavior in the function of the component should be mitigated by safety requirements to ensure that the system is safe.

The primary contribution of this work is a method to augment STPA with an error ontology described using formal notations and expressions. This will allow us to identify the unsafe behavior of a component based on certain criteria for existing application examples, build expressions to specify input / output / functionality for each component in the feedback control loop, identify unsafe functional behavior finding more hazardous possibilities using the error ontology and identify mathematical expressions for each error flow. The augmented method is used to help stakeholders or safety analysts during the hazard analysis process. Formal notations and expressions are created in the canonical feedback control loop architecture and are used to find additional unsafe possibilities in the operational system context. Finding hazardous possibilities before development of the system improves confidence that the system will behave correctly when it faces hazardous conditions.

**Background**

In this section, we give background on current hazard analysis approaches and error ontologies that provide information about error types.
**Systems-Theoretic Process Analysis (STPA):** STPA is a top-down hazard analysis approach built on the Systems-Theoretic Accident Model and Process (STAMP). The major idea behind this approach is to investigate an accident before it occurs. The main goal is to identify potential causes of accidents, that is, scenarios that may lead to losses, so they can be controlled or eliminated in the system design or operations before damage happens. In a nutshell, it provides scenarios to control and mitigate the hazards in the system design. The method consists of four steps to provide scenarios (Abdulkhaleq and Wagner, 2013):

1. The stakeholder establishes fundamental analyses to identify accidents and the hazards associated with those accidents.
2. The stakeholder designs a feedback control loop for the system to identify major components such as sensors, controllers, actuators, and the controlled process.
3. The stakeholder identifies unsafe control actions that could lead to hazardous states. The stakeholder can use the control table to identify unsafe control actions and can translate it to corresponding safety constraints.
4. The stakeholder identifies causal factors for the unsafe control actions. The safety analyst determines how each hazardous control action could occur by identifying the process model variables for the controller in the feedback control loop and analyze each path to find out how each hazard could occur.

**Error Ontology:** The error ontology, defined in the Error Annex for AADL, represents error types in a hierarchical structure to support hazard analysis. It provides the concept of error type to characterize the types of errors to be propagated. It presents an error event for an activated fault type and presents an error behavior state for each failure mode type. The error type can be described as a categorical label to characterize the type of error declarations in error propagations, error events, error behavior states, error flows, and error containment. Also, the label is used to characterize condition declarations for state transitions, detections, and outgoing error propagations. The stakeholders can use error types to describe how the components could fail and to associate with error events. For instance, the effect of a sensor failure might be that it dispatches an incorrect reading (value error), it misses a reading (item omission), or it does not provide any readings (service omission). These effects can be caused by different factors like overheating, radiation, and low power. The error ontology classifies errors into six major error types (Feiler and et-al, 2016):

1. **Service Errors:** Represent errors which are related to delivering service for items. Service errors can be categorized as omission errors, which represent no service delivered for the items (such as loss of a message or command), and commission errors, which represent unexpected service provided for the items (such as unintended incoming data).
2. **Value Errors:** Represent errors which are related to value domain of a service. Value errors can be categorized as value errors for individual service items like out of range sensor reading, value errors for sequences of service items like bounded value change, and value errors related to the service as a whole like out of calibration.
3. **Timing Errors:** Represent errors which are related to the time domain of a service. Timing errors can be categorized as individual service items like rate errors, timing errors for sequence of service items like early/late item delivery, timing errors related to the service as a whole like early/delayed services.
(4) **Replication Errors:** Represent errors which are related to delivery of replicated services. For example, replicated service items delivered for one recipient or to multiple recipients.

(5) **Concurrency Errors:** Represent errors which are related to behavior of concurrent systems like executing tasks concurrently to access shared resources. Errors are distinguished between race condition errors and mutual exclusion errors.

(6) **Access Control Errors:** Represent errors which are related to operation of access control services like authentication and authorization errors.

### Architecture Discussion

The heart of safety-critical systems is a feedback control loop architecture used to analyze the safety of the system. We wish to improve the safety of such systems by introducing a more thorough safety analysis for system development. Figure 1 shows the feedback control loop annotated with expressions written in the formal notation. The figure consists of four component types: sensors, controllers, actuators, and controlled processes. These components have incoming and outgoing variables used to send and receive information. For example, measured variables (Vme) are used to send information from the sensor to the controller in the form of measured values of the attributes, control actions (CA) are the output of the controller and can be considered a command to the actuator, manipulated variables (Vma) are the output of the actuator and are used to control the behavior of the controlled process, controlled variables (Vc) are the output of the controlled process and are used to monitor the actual behavior of the controlled process and update the information read by the sensor.

In addition, the components have functions to provide a relationship among input and output variables: **First,** sensor functions (Fs(Vc)=Vme) can provide a relationship among controlled and measured variables, reading values of the attributes and updating the values. **Second,** controller functions (Fc(Vme, FPM(Var, Val, Cond)) = CA) describe the relationship among measured variables (Vme) as an input to the controller and control actions (CA) as an output of the controller. The controller functions can be used to take input from the sensor, process it, and make a decision to give a command to the actuator. The controller has a mental model function, also known as the process model function (FPM), which can be used to determine what control action is needed based on receiving the content of the measured variables. The mental model function (FPM(Var, Val, Cond)) has three important elements: (Var) is the set of variables which are used by the process model, (Val) is the set of values that can be used by the variables and (Cond) is the set of conditions that can be used by the process model to make a decision. The conditions are the values expected for the measured variables by the process model function. The output of the controller (i.e CA) depends on the process model decision, or the condition, where: { ∀CA: CA→True iff Cond→True} means the control action is safe if and only if the condition is true. Otherwise, the control action is inadequate and can become an unsafe action. Also, in the controller we have some relationship among variables such as: {Vme ∈ Var} means measured variables of the sensor are one of the elements defined by the process model variables, {Var, Val} ∈ Cond} means the process model make a decision based on the defined variables and their values, and {FPM ⊆ Fc} means the process model function is a part of the controller function. The controller has a duration condition { ∀CA: TCA={MinTCA, MaxTCA}} which can be described as the minimum and maximum time for each duration of control action (TCA). **Third,** the actuator function (FA(CA)=Vma) can build a relationship among the control actions and manipulated variables to follow controller instructions and execute the commands. **Fourth,** the controlled process function (FCP(Vma)=Vc) can make a
relationship among manipulated variables and controlled variables to implement the controller's decision and update the information of the sensor through the controlled variables. **Fifth**, we have a faulty component identifier ($INTF$) to show that the component will deliver incorrect information to the next component and the faulty component will become a source of error propagation, where $\{INTF \in DC\}$ means ($INTF$) is in the set of detection conditions ($DC$). Also, the result of the faulty component is an error ($Err$). The error has an effect on the component's behavior. If the error propagates to the other components, it may or may not lead to unsafe interaction among components, and it may or may not cause hazards. $\{Err \in EO\}$ means ($EO$) is in the AADL error ontology.

**Method Discussion**

We extend each step in STPA with the error ontology described in a formal notation. The steps are:

1. **Specify criteria to identify the unsafe behavior of a component:** This step involves specifying an identifier to represent a faulty component in the feedback control loop that propagates an error to the next component. In this step, we have $INTF$ as a faulty component identifier and $Err$ as a result of the fault.

2. **Build notation to specify input/output as well as functions for each component:** The purpose of this step is the construction of the feedback control loop with mathematical notation and expression for describing the functional behavior of each component as shown in Figure 1. In this step, we have input and output variables for each component such as $Vc$, $Vme$, $CA$, and $Vma$. Also, we provide a specific function for each component to build a relationship among input and output variables such as $FS$, $FC$, $FPM$, $FA$, and $FCP$. 

![Figure 1 — Feedback control loop augmented with mathematical notations and expressions](image-url)
3. **Identify unsafe functional behavior and find more hazardous possibilities using the error ontology:** This step helps to identify unsafe functional behavior of a component that violates safety requirements for the associated overall system. For example, the internal failure of a component can give incorrect results to the component leading to the function of that component potentially becoming an unsafe function. In addition, we use the error ontology described in a formal notation to find the possibilities of unsafe functional behavior which leads to the system behaving incorrectly. Then, we identify the corresponding safety constraints which are needed to mitigate the effect(s) of the unsafe functional behavior.

4. **Identify mathematical notation for each error flow:** This step assists in identifying a formal expression for each error flow in the feedback control loop architecture. Based on our previous work (M. Rashid and McGregor, 2017), we know the propagated error cuts across three components based on a three-way interaction format. For that reason, each error flow should contain three functions. In this case, the proposed safety constraints should cover three components to mitigate the effect of the error flow.

**Example:** We take the automated train door controller system, as described in (Thomas and Leveson, 2011), as an example. We have the following steps to find more hazardous possibilities:

1) Train doors are locked. The train driver is unable to open the door because the sensor provides incorrect readings to the controller. This leads to the passengers being stuck inside the train when the train is stopped at the station platform.

\[
V_{me} \text{ if Measured Values } \in V_{me} \\
F_S(V_c) = \begin{cases} 
\text{Err} & \text{if Measured Values } \notin V_{me}, \text{ where } F_S(\text{INTF}) \rightarrow \text{Err}, \text{ Err } \in EO 
\end{cases}
\]

In (eqn. 1), we express the behavior of the sensor when it sends the correct measured values to the controller via measured variables. The controller is able to recognize the values that were previously defined in the variables. But, in (eqn. 2), we express a faulty sensor; it will send different values to the controller. The controller, then, is not able to understand those values leading to the stuck door.

2) The following identified functions are used to express the response of the controller for the faulty sensor expression.

<table>
<thead>
<tr>
<th>Function</th>
<th>Sensor Function</th>
<th>Controller Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Door</td>
<td>( F_S(V_c, \text{INTF}) = \text{Err} ), ( \text{if Err } \in \text{ErrValues} )</td>
<td>( F_C(F_{PM}(\text{Var}, \text{Val}, \text{Cond})) \neq CA ), ( \text{if ErrValues } \notin \text{Val}, \text{ Then Cond } \rightarrow \text{False} )</td>
</tr>
</tbody>
</table>

As shown in Table 1, the function of the sensor shows the internal failure of the component and provides incorrect information to the controller (i.e. the output of the sensor is an error (Err) which can be considered as one of the errors in the AADL error ontology [such as Error Values]). The function of the controller shows that the incoming error value is not an element in the defined values of the process model function. For that reason, the decision of the process model is going to be incorrect. This affects the output of the controller
because the condition of the decision is false. Thus, the output of the controller is not correct and could lead to a hazard.

<table>
<thead>
<tr>
<th>Possibilities</th>
<th>Description</th>
<th>Is this possibility hazardous?</th>
</tr>
</thead>
<tbody>
<tr>
<td>If ErrValue &gt; MaxVal</td>
<td>The controller does not open the door when it gets above of the range values.</td>
<td>Yes</td>
</tr>
<tr>
<td>If ErrValue &lt; MinVal</td>
<td>The controller does not open the door when it gets below of the range values.</td>
<td>Yes</td>
</tr>
<tr>
<td>If ErrValue $\not\in$ Val</td>
<td>The controller does not open the door when it gets none of the defined values.</td>
<td>Yes</td>
</tr>
<tr>
<td>If ErrValue = &quot;Null Value&quot;</td>
<td>The controller does not open the door when it gets omitted error value.</td>
<td>Yes</td>
</tr>
<tr>
<td>If ErrValue = &quot;Stuck Value&quot;</td>
<td>The controller does not open the door when it gets the same repeated value.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

In Table 2, we show the possibilities of error values from the faulty sensor function to the controller function. We show an expression for each possibility and have a description for it. These possibilities are hazardous because the controller does not send the command to open the door due to receiving different values from the sensor function.

3) How the faulty sensor can impact the door actuator function?

As shown in Table 3, the flow expression demonstrates the internal failure of the sensor which effects the output (i.e., manipulated variables) of the actuator. The sensor propagates an error to the actuator through the incorrect action of the controller. For example, when the door controller receives abnormal values from the sensor, the controller will do incorrect computations and sends incorrect commands to the actuator. This will lead to the actuator executing incorrect actions; it can be seen then as an error in the output of the actuator. To solve this problem, we need to provide a safety constraint to mitigate the effect of the error on the actuator which comes from the sensor. In this case, we need to build a specific safety constraint based on the three-way interaction format to cover the flow. For that purpose, we build $SC(S, C, A, Err)$ which means the controller ($C$) is required to provide a control action for the actuator ($A$) with respect to the error ($Err$) context for the message that comes from the sensor ($S$).

**Method Implementation**
In this section, we implement the method described in previous section, Architecture Safety Analysis Method (ASAM). It is used for developing safety-critical systems, that will allow system stakeholders or safety analysts to mitigate the effects of the errors through safety constraints. Generally, in the method, we focus on finding safety constraints and satisfy them within the system model. Specifically, we inject the AADL error ontology into the architecture model to identify the unsafe control actions and causal factors of the unsafe control actions. We satisfy the safety constraints that we have found within the system model to ensure the hazardous situations do not occur. For that purpose, we focus on introducing compositional reasoning in our method to ensure that the unsafe control action will not happen.

From a technical perspective, ASAM is a new software safety analysis tool which works with AADL models annotated with an annex (i.e. asam annex) and supported by the Open Source Architectural Tool Environment (OSATE). It is used to analyze and generate a report based on information attached to each component in the feedback control architecture model. In fact, ASAM provides several statements to feed the method's steps into the system's architecture model such as error statements, error propagation statements, internal failure statements, safety constraint statement that either handle or prevent hazards. The goal of the statements is to not let errors propagate from one component to another. If it is propagated, ASAM can handle incoming errors and prevent outgoing errors via the implementations of the safety constraints. ASAM lets the safety analyst record the severity level of the hazard for the specific error type based on the probability of occurrence for that type of error. Also, ASAM satisfies the safety constraint statements to ensure that the error effect or unsafe action has been mitigated.

**Example:** What will happen if the sensors in an adaptive cruise control (ACC) report incorrect values for speed and distance of the car in front of its car during driving because of an internal failure?

![Figure 2— ASAM's results for ACC system example](image)
We divide Figure 2 into four steps. The numbers in Figure 2 correspond to the steps shown below:

(1) The ASAM annex allows stakeholders to record the specific information in the implementation part of each component in the ACC architecture model. For example, we have recorded the following information: error types for the sensor's internal failure (above range of speed and below range of distance), unsafe control action for each error type, general or specific causes for each unsafe control action, probability of occurrence for each error type, severity level of hazards for each error type, and safety constraints to mitigate the effects of the error types. Also, ASAM allows the safety analysts to record the particular information in the specification part of each component to verify the information have been recorded in the implementation part. For example, in the ACC model, we need to ensure the system does not apply the brake when it has incorrect values of speed or distance.

(2) A feedback control loop has been built for the ACC system. Figure 2 shows error flow for each error type based on the three-way interaction format. Figure 2 shows two error flow: one for distance error and the other for speed error. The first error flow (i.e., red line) shows the error propagation from sensor to actuator. The error in measuring of distance propagates to the controller through the output data port. The error distance value in the sensor can be prevented from being communicated through a safety constraint statement. But, if it is not prevented, it propagates to the controller. The controller also can handle the incoming error distance value through in data port according to the safety constraint. If the controller is not able to handle that, it sends an inadequate command to the actuator. Then, the actuator will execute an inadequate action. The effect of that error on the manipulated variable which is the output of the actuator. The second error flow (i.e. blue line) has the same description.

(3) The analysis report shows the source of faults in the ACC model, which is the sensor. Each fault in the sensor produces errors. Each error has a chance to harm people based on the probability of occurrence for that error. ASAM compares the recorded probability value with probability of occurrence thresholds to find the probability of which hazard could occur. For example, an "above range of speed" error results in a critical level and a "below range of distance" error results in a catastrophic level. Each error in the sensor gives an unsafe control action to the controller which results in sending an inadequate command to the actuator. For example, the ACC controller warns the driver to apply the brake. If the driver does not do that action, the system will automatically do it. Also, general and specific causes for each unsafe control action have been identified. Finally, ASAM satisfies the safety constraints that have been recorded to mitigate the effect of the errors. For example, the ACC system verifies that should not apply the brake when it has incorrect values of distance and speed.

(4) The report shows three internal failures in the ACC model. We have a description for each one: First, the sensor has an internal failure, it gives two errors to the sensor. The sensor has a safety constraint to mitigate the effect of the first error (i.e., above range of speed error). But, the second error (i.e., below range distance error) propagates to the controller because the sensor does not have a safety constraint for it. For that reason, the controller mitigates the effect of the sensor's error. Second, the controller itself has an internal failure, it gives two errors (above range of speed and below range of distance) on the output data port. Those two errors are propagated to the actuator because the controller has a safety constraint to handle incoming errors only. But, the actuator is able to handle both
controller’s errors. Third, the actuator has an internal failure, it gives an error to the actuator, but the actuator is able to prevent it being propagated out to the controlled process.

**Conclusion and Discussion**

It can be concluded that ASAM is different than STPA because it provides mathematical notation/expression and error ontology for the feedback control loop architecture to find more safety constraints during hazard analysis. This allows stakeholders to use formal methods to find more hazardous possibilities and mitigate them using safety constraints.

ASAM is different than STPA in finding more hazards using the three-way communication format as opposed to using the two-way format. This allows the identification of potential internal failures of the major components in the feedback control loop architecture. The result of each internal failure is an error. The error propagates and cuts across components based on the three-way interaction format. This allows stakeholders to identify a new unsafe control action and allows the reduction of the potential effects of residual hazards in the operational system context.

ASAM is different than STPA in using verification procedures to verify that each hazardous condition cannot occur. ASAM verify the safety constraints against the system model with injected errors. The output of the automatic model checking identifies the behavior of the system. This allows us to determine that either unsafe behaviors occur (which means the error leads to the hazardous condition in the system) or verify that the error does not lead to unsafe behavior of the system.

All in all, ASAM allows system stakeholders to find new hazards and new safety constraints that STPA does not find. This does not mean that our method is a replacement for STPA; however, it supports more effective and rigorous analyses of the elements of safety-critical systems.

**References**


